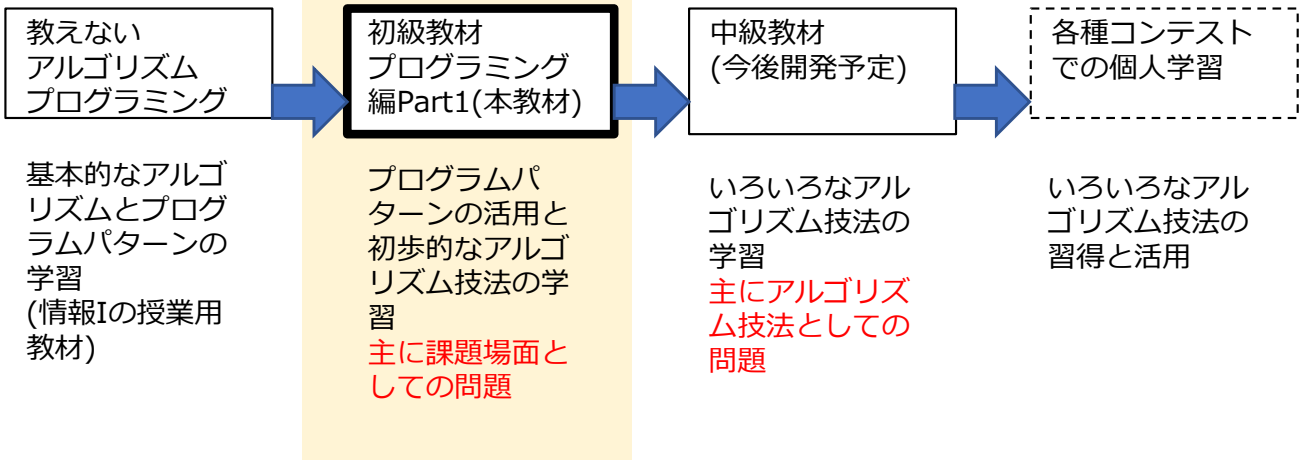
2025年大学入試テスト
情報科　対策教材

アルゴリズムとシミュレーション
初級教材 プログラミング編Part1

使い方や教材の考え方のビデオ
https://youtu.be/G4NPpFdfn3A

2022年11月
ドラフトVer0.9
太田 剛

教材の位置づけ



教材内容

B01 Python基礎(文字列操作):ランレングス圧縮
B02 Python基礎(while): 金利計算(プログラミング版)
B03 Python基礎(乱数):モンテカルロ法
B04 Python基礎(二次元リスト): 二次元リスト検索
B05 Python基礎(関数, 配列操作):素数を求める
B06 Python基礎(再帰):フィボナッチ数
B07 Python基礎(文字コード):シーザー暗号
B08 Python基礎(論理演算):論理回路
B09 Python基礎(if-elif-else):Fizzbuzz
B10 文字列操作:文字削除
B11 合計の計算(文字列処理)
B12 10進数→2進数変換
B13 硬貨での支払い枚数(貪欲法)
B14 学園祭の見学(区間スケジュール)
B15 グループ分け
B16 パリティチェック(誤り検出/訂正)
B17 感染症の計算(微分計算:SIRモデル)
B18 ファーストフードの待ち時間(待ち行列)
B19 Robot操作(ロボット平面移動)
B20 最短経路(グラフ)

Python基礎(文字列操作):ランレングス圧縮

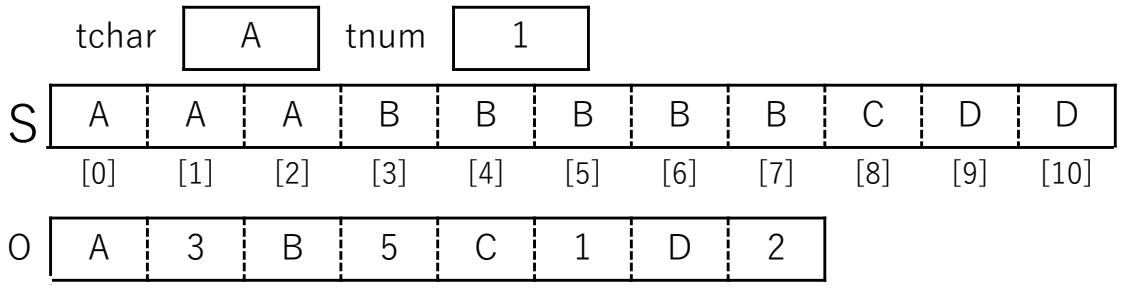
サイズの大きなデータは送信したりファイルに保存する場合、そのサイズを小さくする場合に、圧縮することがある。　ここでは、文字列の基本的な方法としてランレングス圧縮を使用してみる。この圧縮方法は、文字が連続する場合に、繰り返しの数に置き換えるものである。ここでは、例えば

AAABBBBBCDD という文字列を
A3B5C1D2という文字列に置き換える

方法としては、文字が連続する場合、その文字一文字の後に繰り返しの数に置き換える。

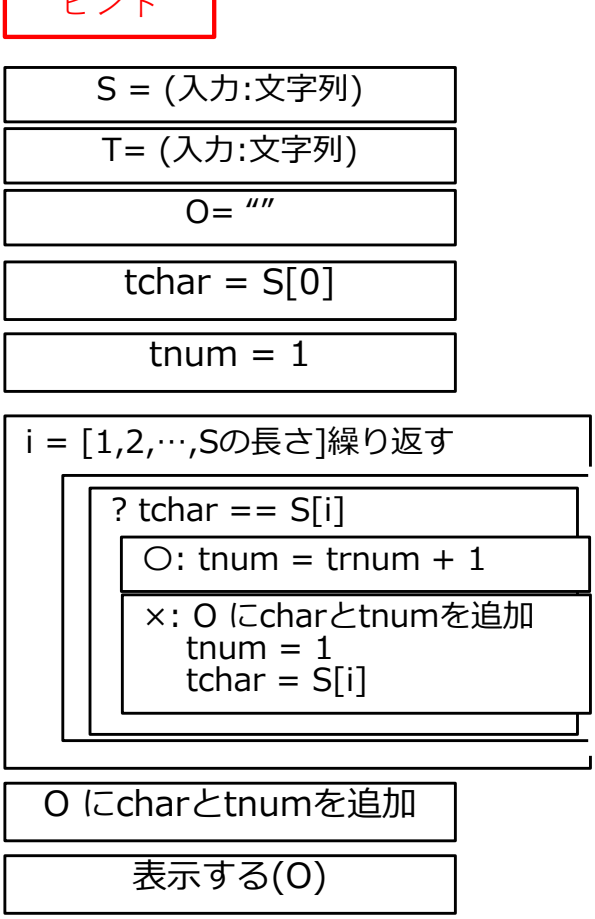
入力した文字列をランレングス圧縮して、その結果を表示するプログラムを作成する。

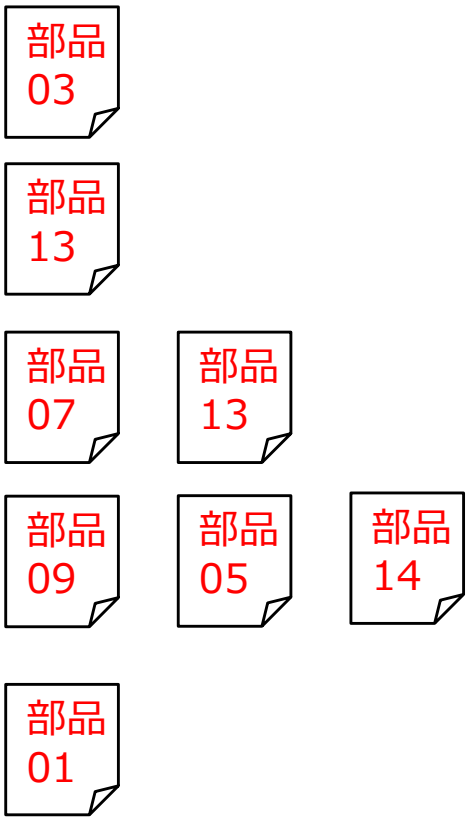
考え方



① 初めて出てきた文字をtcharに入れて、tnumを1にする。
　(プログラムの初めにはリストSの先頭の文字を入れておく。
② 同じ文字が続いている場合は、tnumの数を増やす。
　違う文字が出た場合は、tcharの内容とtnumの数をリストOに追加する。
③ ②の作業をリストSの文字列の長さだけ繰り返す。
④ 最後に残っているtcharの内容とtnumの数をリストOに追加する。

ヒント





Python基礎(while): 金利計算(プログラミング版)

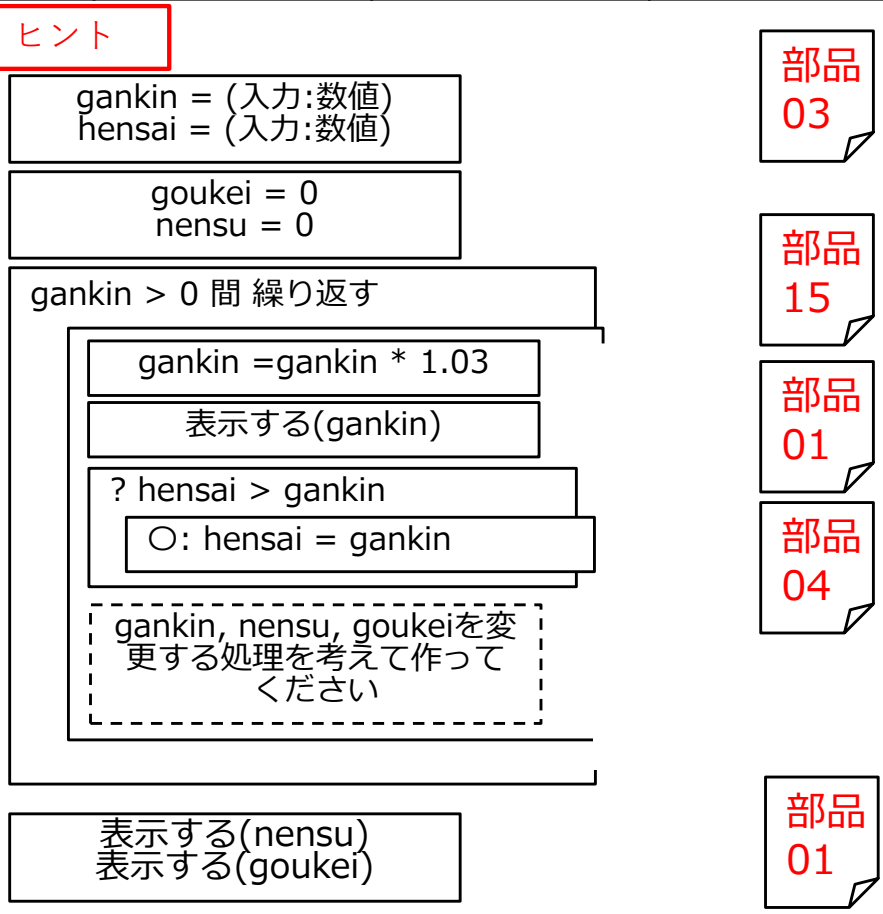
お金を借りた場合のローン計算をシミュレートして、返済年数と総返済額を表示するプログラムを作成する。課題は下記のように単純化している。
返済は毎年、借入金と利息分を毎年均等に(同じ金額)で返済する方法を使用する。この課題では、年利を0.03(3%)としして、借入金(元金)と毎年の返済金を入力すると、毎年の借り入れ金を表示して、最後に返済年数と総返済額を表示する。
(ローン計算の考え方としては、かなり簡略化しています)

考え方

例えば、初めに、1,000,000円借りて、毎年200,000円返済する場合
・毎年元金に利息を加えたものが新しい元金になる。そして、200000円返却したものが、その時の元金残高になる
・次の年は、前年度の残高が新しい、元金となり、計算する。
・返済金200000円より、元金+利息が小さい場合は、元金+利息を返済して、元金が0より大きい間は、支払いを続ける。

	元金	元金+利息 (元金 * 1.003)	返済金	元金残高
1年目	1000000	1030000	200000	830000
2年目	830000	854900	200000	654900
3年目	654900	674547	200000	474547
4年目	475747	488783	200000	288783
5年目	288783	297446	200000	97446
6年目	97446	**100370**	**100370**	0

ヒント



Python基礎(乱数):モンテカルロ法

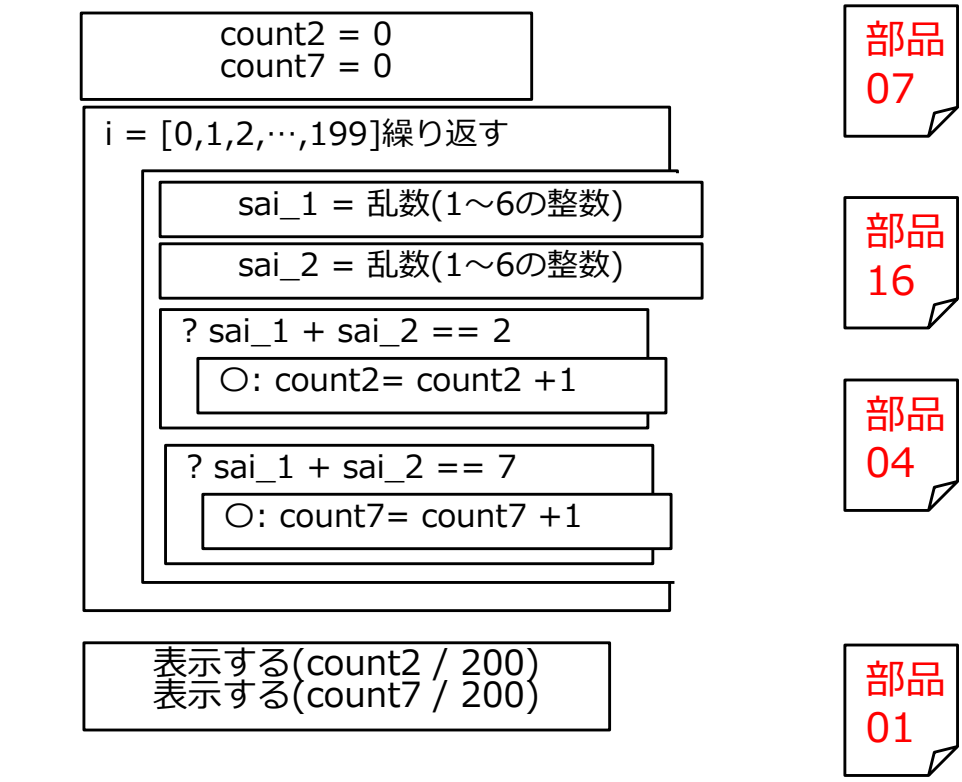
1～6までの目のサイコロを2個使用して、同時に200回振って、2個のサイコロの目の合計が、それぞれ2と7になる確率を表示するプログラムを作成する。

考え方

sai_1	A	sai_2	1	count2	5	count7	6

サイコロを振って、ランダムな数を作るのにPythonの標準ライブラリーのrandomモジュールを使用する。
① 以下の②～④の処理を200回繰り返す。
② sai_1とsai_2に1～6までのランダムな数を入れる
③ もし、sai_1とsai_2の合計が2ならば、count2を1増やす
④ もし、sai_1とsai_2の合計が7ならば、count7を1増やす
④ 最後にcount2とcount7を200で割って、確立を表示する。

ヒント



Python基礎(二次元リスト): 二次元リスト検索

コードを入力すると、対応する都道府県の名前を表示するプログラムを作成する。
また、対応するものが無い場合は、該当なしと表示する。
なお、情報は下記のような二次元のリストに入っているものとする。

C01	北海道
C05	秋田県
C13	東京都
C26	京都府
C30	和歌山県

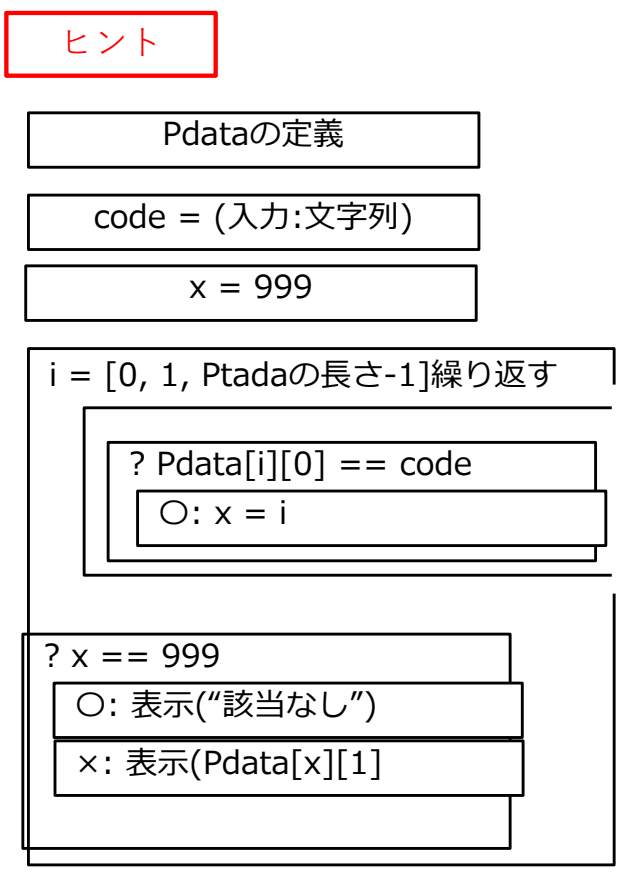
pythonでの二次元リストの定義
Pdata = [[“C01”, “北海道”],
[“C05”, “秋田県”],
[“C13”, “東京都”],
[“C26”, “京都府”],
[“C30”, “和歌山県”]]

考え方

code [　　]　　x [　　]

① 配列の何番目になっているか入れるためxの変数に0を代入しておく。
② 探すコードのcodeに入力する。
③ ④の処理を、リストの長さ繰り返す。
④ もしcodeとPdadaのコードが一致すれば、xに繰り返しの番号を入れる
⑤ 最後iにxの内容と999を比べて、同じであったら”該当なし”と表示し、そうでなければ、Pdataのxの番号の都道府県名を表示する。

ヒント

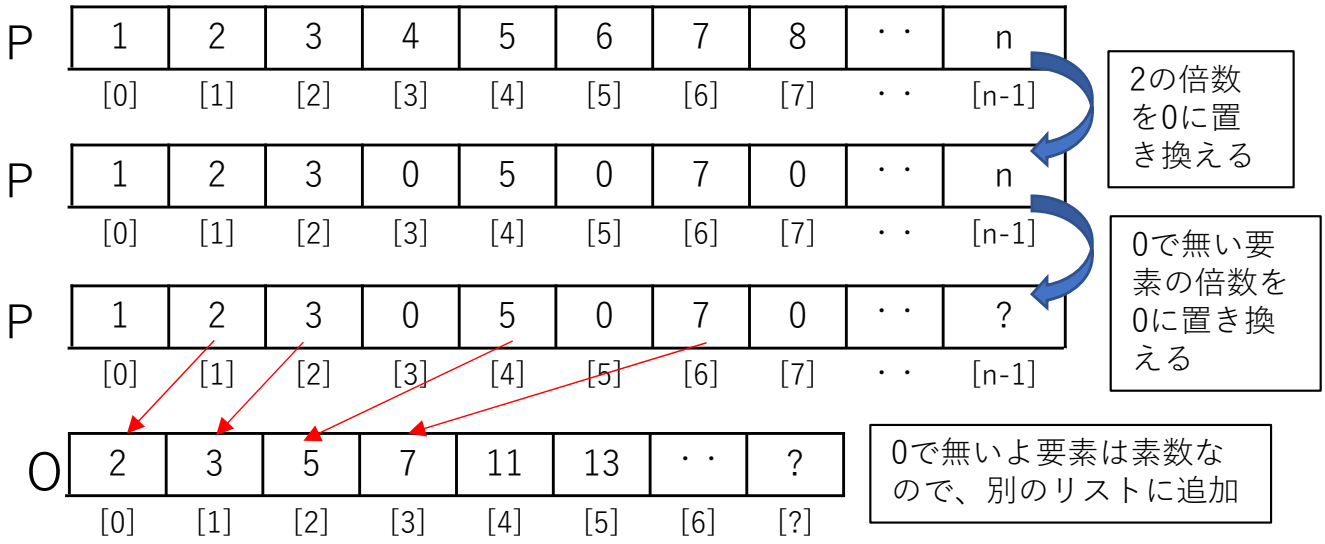


部品 17
部品 03
部品 07
部品 04
部品 17
部品 05

Python基礎(関数, 配列操作):素数を求める

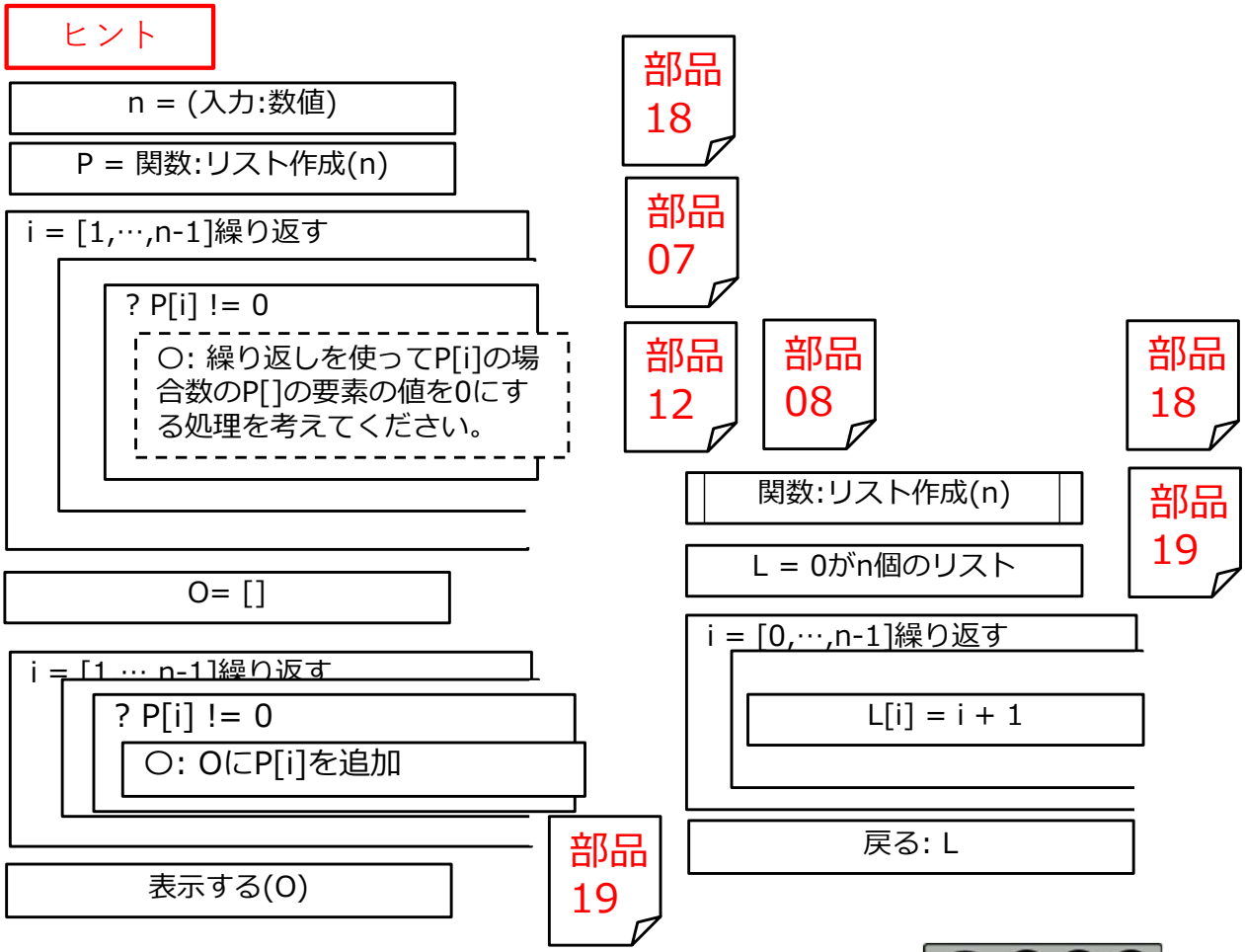
数値nを入力して、n以下の素数を表示するプログラムを作成する。

考え方



① 入力したnまでの数が入ったリストPを作成する(この課題では関数で作成する)
② 2～n/2まで③の処理を繰り返す
③　Pの対応する数が0でなければ、その数の倍数を0に置き換える
④Pで2以上で0で無い数をリストOに追加していく。
⑤ Oをょう字する

ヒント



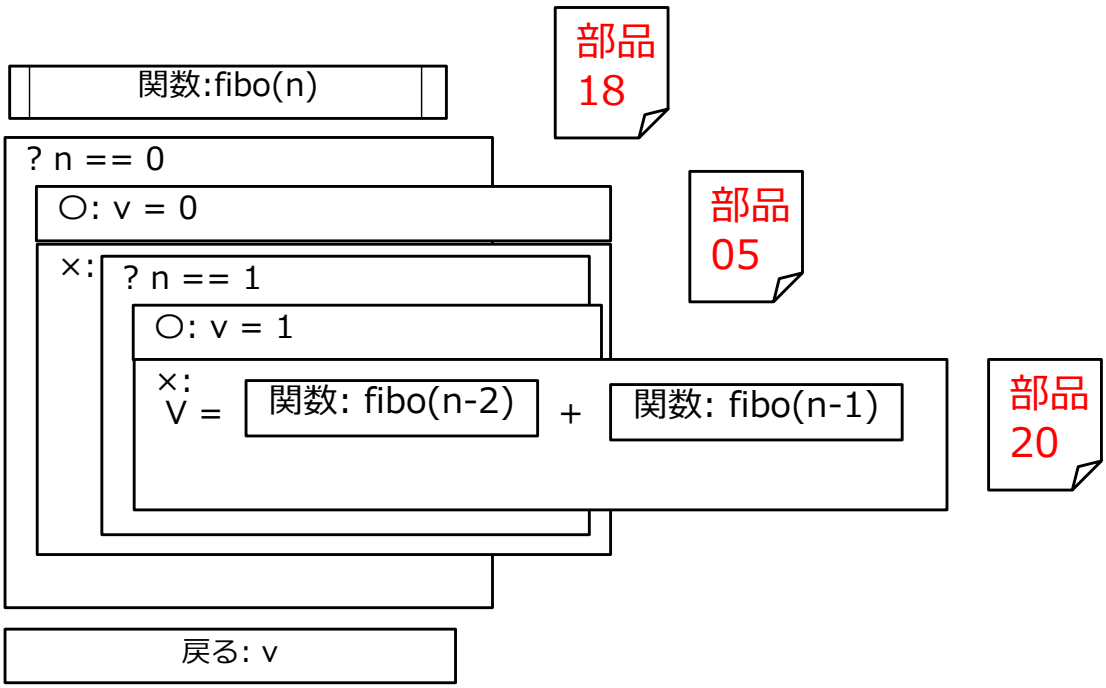
Python基礎(再帰):フィボナッチ数

フィボナッチ数は0, 1, 1, 2, 3, 5, 8, 13, 21のように、となりあった数の和が次の数となるような数列であり、nを入力して、そのn番目の数を求めるプログラムを作成する。なお、引数をnとするフィボナッチ数を求める関数をプログラムの中で作成する。(はじめの数は0番目とする)

考え方

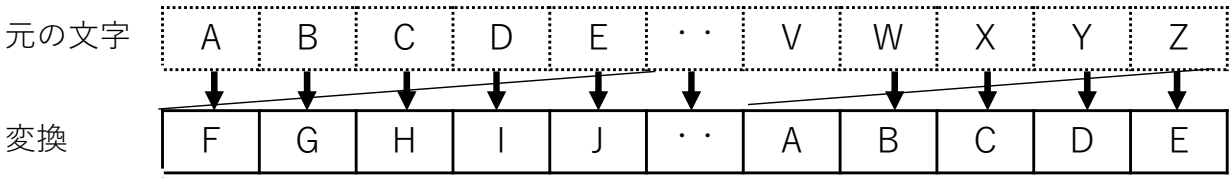
n番目のフィボナッチ数求める関数をfibo(n)とすると、
fibo(0) = 0, fibo(1) = 1, fibo(n) = fibo(n-1) + fibo(n-2)と定義される。
プログラムでfibo(n)の関数を定義して、fibo(n)内で、fibo(n-1), fibo(n-2)を再帰的に呼び出すことで、fibo(n)を求める。

ヒント



Python基礎(文字コード):シーザー暗号

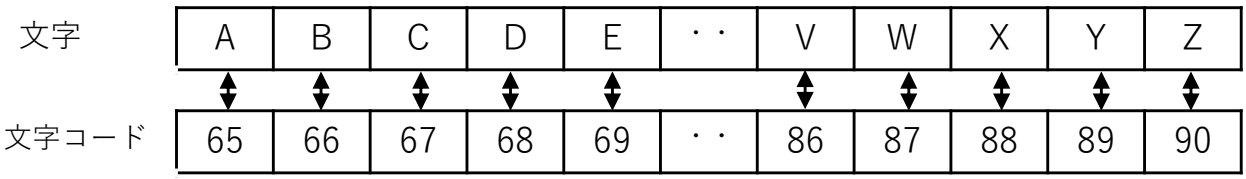
シーザー暗号は、各アルファベットをn文字シフトした文字に変換する古墳的な暗号の手法である。例えば、5文字右にシフトする場合の文字の対応は次のようになる。



上記の5文字右にシフトする方法で、入力した文字列を暗号化して表示するプログラムを作成する。なお、入力する文字列はA～Zの大文字のアルファベットとする。

考え方

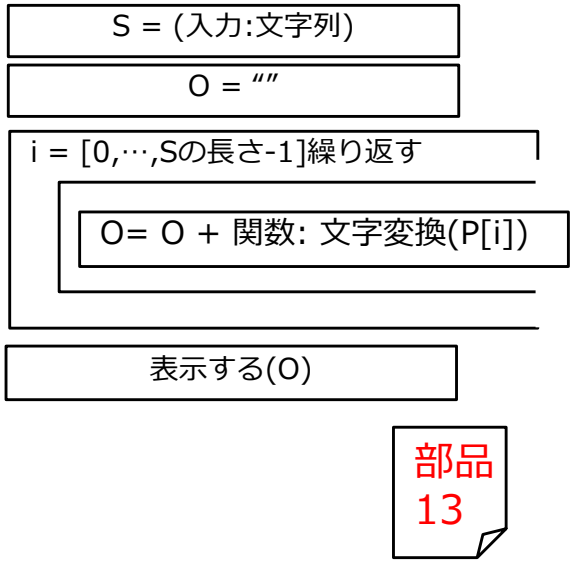
コンピュータ内部で文字を扱う場合は、各文字に対応して、バイト単位の数値が割り当てられて処理される。その文字と数値の対応を文字コードと呼んでいる。文字コードにはいろいろな種類があるが、例えば、ASCII文字コードでは、数のような対応になっている。



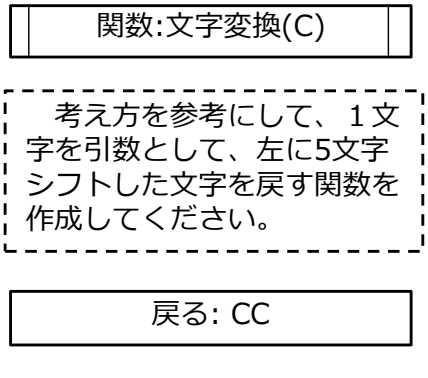
この課題で、5文字右にシフトする場合は、文字コードの数値としては、5加えたものになる。ただし、90を超えた場合は、またAに対応し65に戻して計算する必要がある。なお、Pythonでは、文字から文字コードに変換する場合は、ord()関数、文字コードから文字に変換する場合は、chr()関数を使用する(部品21参照)

プログラムとしては、文字を与えたときに５文字左にシフトした文字を戻す関数を作成して、入力した文字列を格納したリストSの内容を1文字づつ変換する。

ヒント



部品13



部品18

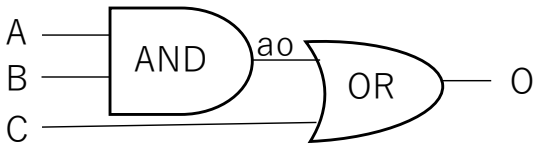
部品21

部品04

Python基礎(論理演算):論理回路

下図のようなANDとORの回路があり、A,B,Cの入力は右図のような組み合わせがある。**A,B,Cのすべての組み合わせに対して出力Oがどのような値になるか表示するプログラムを作成する。**なお、プログラム中ではDで示した二次元のリストで状態を表現する。

A	0	0	0	0	1	1	1	1
B	0	0	1	1	0	0	1	1
C	0	1	0	1	0	1	0	1
O	?	?	?	?	?	?	?	?



pythonでの
二次元リストの定義
D = [[0, 0, 0”],
[0, 0, 1”],
・・・
[1, 1, 1”]]

考え方

①以下の②～④の処理のDの要素だけ繰り返す
② A ==1 かつ B == 1 を判断して、aoの出力に1又は0を設定する。
③ ao==1 又は C==1を判断して、Oの出力に1又は0を設定する。
④ Oを表示する

ヒント

Dの定義

部品17

i = [0, 1, Ptadaの長さ-1]繰り返す

考え方と部品22を参考にして、D[i]の要素である[?,?,?]をもとに、出力結果であるOを求める処理を作成してください。

表示する(O)

部品07

部品05

部品17

部品22

Python基礎(if – elif -else):Fizzbuzz問題

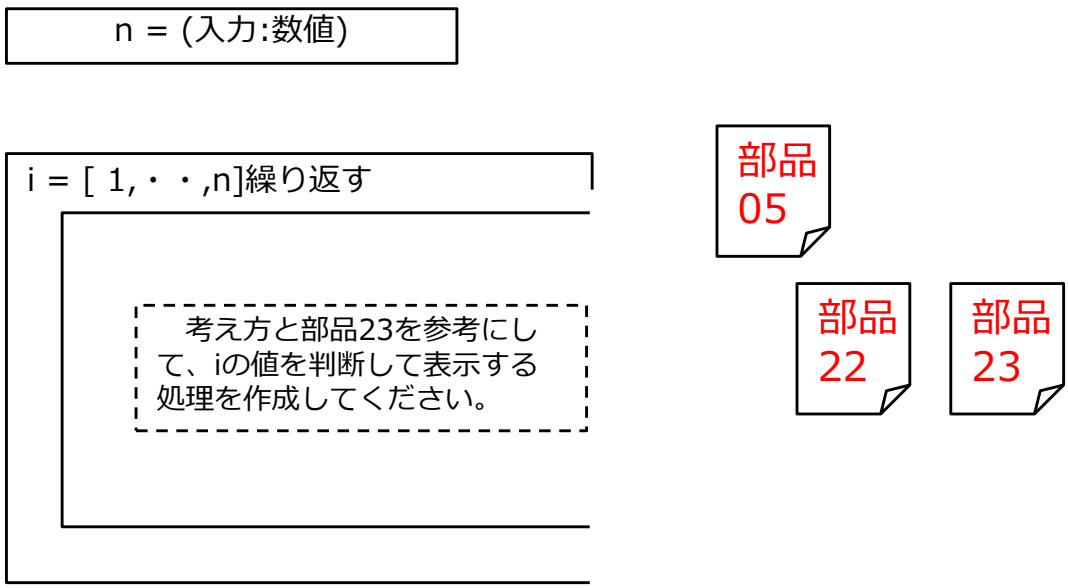
Fizz Buzzという遊びのプログラムを作成する。3の倍数の時は”Fizz”と表示。
5の場合の時は”Buzz”と表示。
3と5の倍数の時は、”Fizz Buzz”と表示。
それ以外は数字を表示する。どの数まで表示するかは初めに入力する。

実行結果例
```
20　←　入力
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
```

考え方

以下②～⑤の処理を1からnまでの数で繰り返します。
② 数が3の倍数かつ5の倍数ならば”Fizzbuzz”と表示。
③ ②でなく3の倍数ならば、”Fizz”と表示。
④ ②～③でもなく5の倍数ならぱ、”Buzz”と表示。
⑤ ②～④でもなく場合は、数字を表示
　倍数の判断は、3又は5で割った時の余りが0であれば倍数。

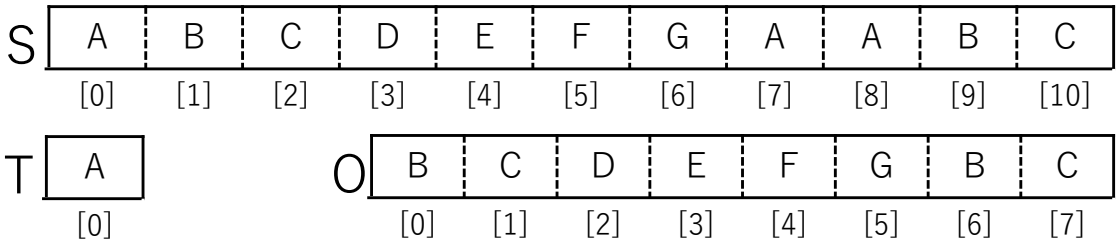
ヒント



文字列操作:文字削除

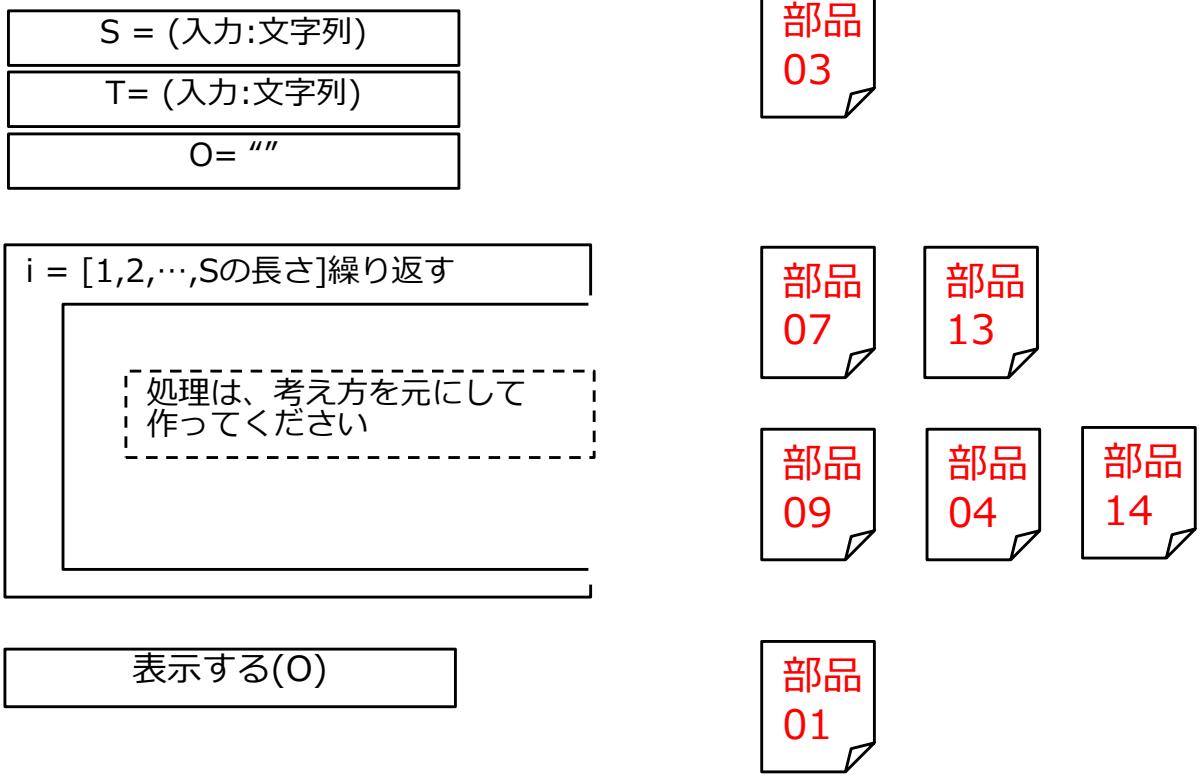
元の文字列を入力して、次に入力して指定した1文字文字を削除するプログラムを作成する。(なお、Pythonには、replaceという関数があるが、これは使用しない)
例えば
ABCDEFGAABC をもとの文字列として入力し、次に Aを削除する文字として入力した場合は、 BCDEFGBCを表示する。

考え方



① 元の文字列をS, 削除する文字をTに入力する。
② Sの文字を初めから繰り返して、Sの文字列の長さだけ繰り返す
　もし、Sのその時の文字が、Tの文字と一致しなければ
　　そうでなければ、Sのその時の文字をOに追加する
③最後にOの文字列を表示する。

ヒント

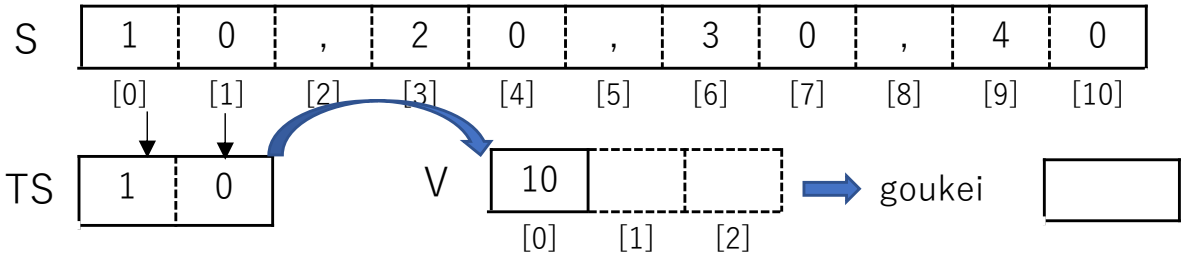


合計の計算(文字列処理と繰り返し)

下図ように、,(カンマ)で区切って入力した複数の数字の合計を計算するプログラムを作成する。なお数字は2個以上入力する。
10,20,30,40　← 入力した数字
100　← プログラムで表示する合計

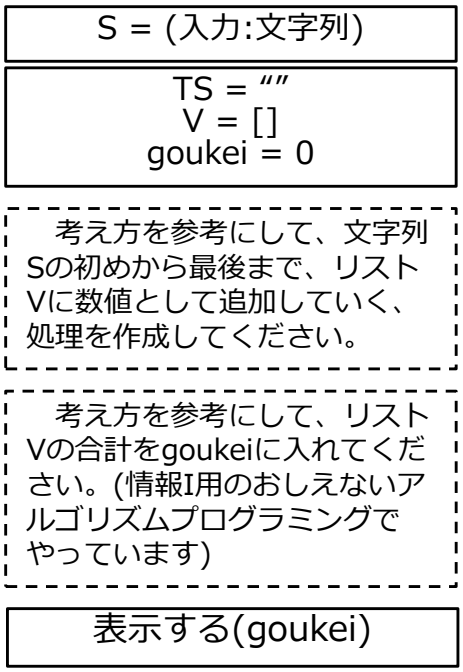
10,100,1000　← 入力した数字
1110　← プログラムで表示する合計

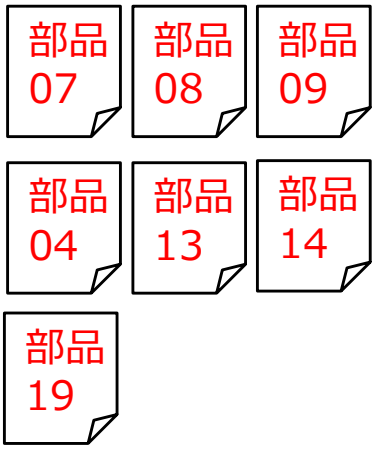
考え方



① 入力した文字をSに入れる。
② Sの文字を初めから見て、,(カンマ)までTSの文字列に入れる。
② ,(カンマ) が出てきたら、TSの文字列を数値にしてリストVに追加する。この時TSはクリアーしておく。
③ Sの文字のチェックが終わったら、TSに残っている文字列を数値にしてリストVに追加する。
④ Vのリストの数値の合計をgoukeiに計算して、表示する。

ヒント





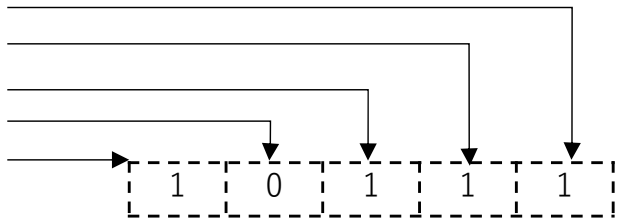
10進数→2進数変換(繰り返し、数値から数字の変換)

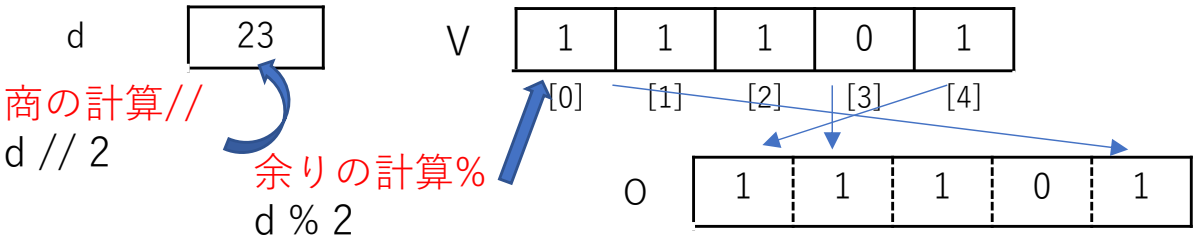
10進数の数値として入力し、2進数に変換して数字(文字列)として表示するプログラムを作成する。

考え方

10進数から2進数の変換は、下記のように、2で割った商が0になるまで繰り返し、各余り、逆に並べると2進数となる。

```
23 / 2 = 11 ・・・ 1
11 / 2 =  5 ・・・ 1
 5 / 2 =  2 ・・・ 1
 2 / 2 =  1 ・・・ 0
 1 / 2 =  0 ・・・ 1
```





① 入力した10進数のdの商(//使用)と余り(%を使用)を求め、商は次の計算用にdに戻し、余りはリストVに追加する。
②①の処理を商が0より大きい間繰り返す。
③リストVに入っている数値を、後ろから数字に変換して、文字列Oに追加していく。

ヒント

```
d = (入力:数値)

V = []
O = ""

考え方を参考にして、dの値の余りをdが0より大きい間、2で割った余りをリストVに追加する処理を考えてください。

i = [Vの長さ-1, Vの長さ-2,… 0]
    繰り返す
    O= O + 関数: str(V[i])

表示する(O)
```

部品 15　部品 13　部品 19

部品 07　部品 08　部品 13

range(m, n, -1)
m > n で増分が負の数の場合
この例では
[m, m-1, m-2, ・・・n+1]の数列を生成する。

硬貨での支払い枚数(貪欲法)

支払い金額を入力したあと、500円、 100円、 50円、 10円、5円、1円の硬貨で支払い枚数が一番少ないなるような、各硬貨の支払い枚数を表示するプログラムを作成する。
各硬貨の枚数は制限がないものとし、硬貨の種類は下記のようにプログラムの中で定義されているものとする。
C = [500,100,50,10,5,1]

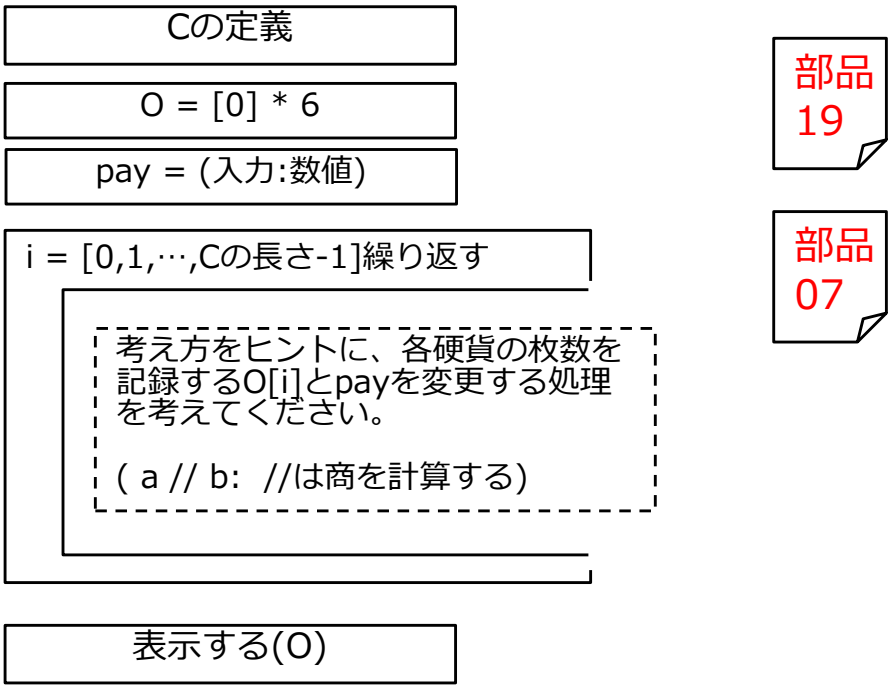
実行例
124　←　入力
[0, 1, 0, 2, 0, 4]

687　←　入力
[1, 1, 1, 3, 1, 2]

考え方

基本的に、大きな金額硬貨から考えて、大きな金額で最大払える枚数を計算して、残りの金額を次の大きな金額で払うことを繰り返す。
今回の問題では、Cに入っている硬貨の大きさが大きい順番になっているので
①　②③の処理をCの要素で繰り返す
②　その時の硬貨で最大支払える枚数を求める
③　金額からその時の硬貨の金額 x 枚数を引いて、あたらしい金額にする

ヒント



学園祭の見学(区間スケジュール)(1)

下図のように、学園祭のイベントA～Iが午後1時から実施される。**一番多くのイベントに参加するには、どのようなイベントを選択したら良いか表示するプログラムを作成する。**

	午後1時からの経過時間(分)														
	20	40	60	80	100	120	140	160	180	200	220	240		開始	終了
A		■	■	■									[0]	20	80
B	■	■	■	■	■								[1]	0	100
C			■	■	■	■	■						[2]	40	140
D						■	■	■	■	■			[3]	100	200
E							■	■					[4]	120	160
F				■	■	■							[5]	60	120
G					■	■	■						[6]	80	140
H										■	■	■	[7]	180	240
I								■	■	■			[8]	140	200

例えば、イベントAは1時+20分から開始して、1時+80分に終了する。プログラムの中では、各イベントのスケジュールは右のように定義されていて、下のような結果を表示する。

[[20, 80], [80, 140], [140, 200]]

```
D = [[20,80],
 [0,100],
 [40,140],
 [100,200],
 [120,160],
 [60,120],
 [80,140],
 [180,240],
 [140,200]]
```

考え方

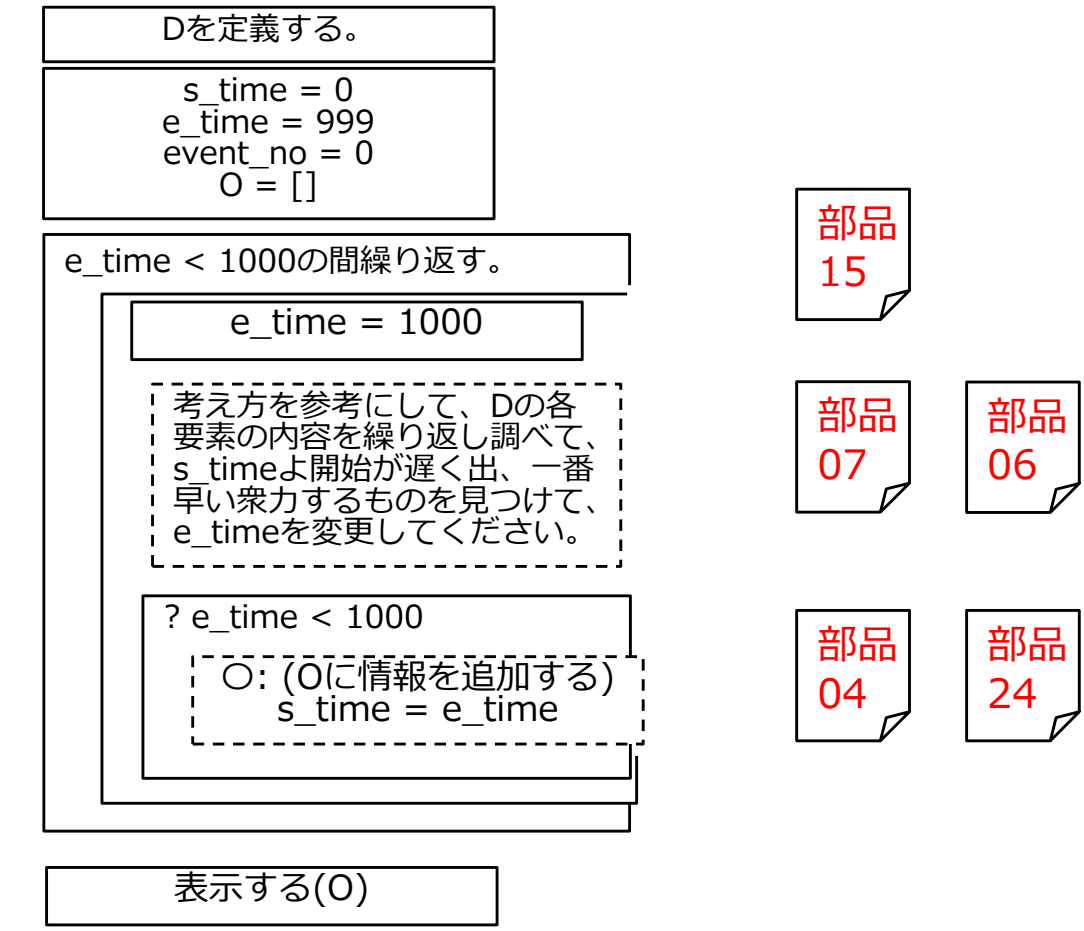
	午後1時からの経過時間(分)														
	20	40	60	80	100	120	140	160	180	200	220	240		開始	終了
A		■	■	→									[0]	20	80
B	■	■	■	■	■								[1]	0	100
C			■	■	■	■	■						[2]	40	140
D						■	■	■	■	■			[3]	100	200
E							■	■					[4]	120	160
F				■	■	■							[5]	60	120
G					■	■	→						[6]	80	140
H										■	■	■	[7]	180	240
I								■	■	→			[8]	140	200

① ② ③ ④

この種類の問題では、とにかく、早く終了するものを順番に選んでいくことで、多くのものを選択できる。
初めの開始時間は、①で、この時、一番早く終了するのはAを選ぶ。
次にAの終了時刻②を新しい開始時刻として、この後に開始され、一番早く終了するGを選ぶ
次にGの終了時刻③を新しい開始時刻として、この後に開始され、一番早く終了するIを選ぶ
次にIの終了時刻④を新しい開始時刻として、この期に開始されるイベントは無いので終了する。

学園祭の見学(区間スケジュール)(2)

ヒント



　考え方で説明した、開始時刻をs_time, 終了時刻をe_time, 最後に表示する要素を加えていくOと、その時の終了時刻が最もイベントの番号を入れるevent_noを定義しいます。

　考え方で示したように、開始時刻を変える大きな繰り返しと、開始時刻の後から始まるイベントで最も早い終了時刻を見つける繰り返しが合わさったものになります。

　このプログラムでは、大きな繰り返しの終了を判断するために、e_timeを特殊な使い方をしています。ここでは、大きな繰り返しの初めで、e_time = 1000(実際のデータにはない大きな数)を入れていて、イベントが見つかった場合は、このe_timeを実際の終了時刻にしています。そのため、イベントが見つからなかった場合はには、e_time = 1000のままなので、終了することになります。そして、初めに、この大きな繰り返しに入るために、e_time = 999を設定しています。**(部品24参照)**

グループ分け

春	0
夏	1
秋	2
冬	3

		優先順番															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
氏名		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
希望順番	1	0	2	0	0	1	0	3	0	1	1	2	2	1	0	2	0
	2	1	1	2	2	0	3	1	2	2	2	1	0	0	2	1	2
	3	2	0	1	1	2	2	0	3	0	0	3	1	3	1	0	1
	4	3	3	3	3	3	1	2	1	3	3	0	3	2	3	3	3

上図のように春夏秋冬の植物を調べる4つのグループを作ることにした。16名の生徒A～Pまでにどのグループに入りたいか希望順番を聞いたみた。**この希望順番を元に生徒を4つのグループに分けるプログラムを作成する。**なお割り当てはクジで決めた優先順番の小さい番号の人から優先的に希望のグループに入れるものとする。なおプログラムでは、生徒の名前は次のようなリストNamaeに格納している。

Namae = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P"]

また、各生徒の希望順番は、右図のようにKibouリストに格納している。実行した時には下記にようにグループの氏名が表示される。

['ACDF', 'EIJM', 'BHKL', 'GNOP']

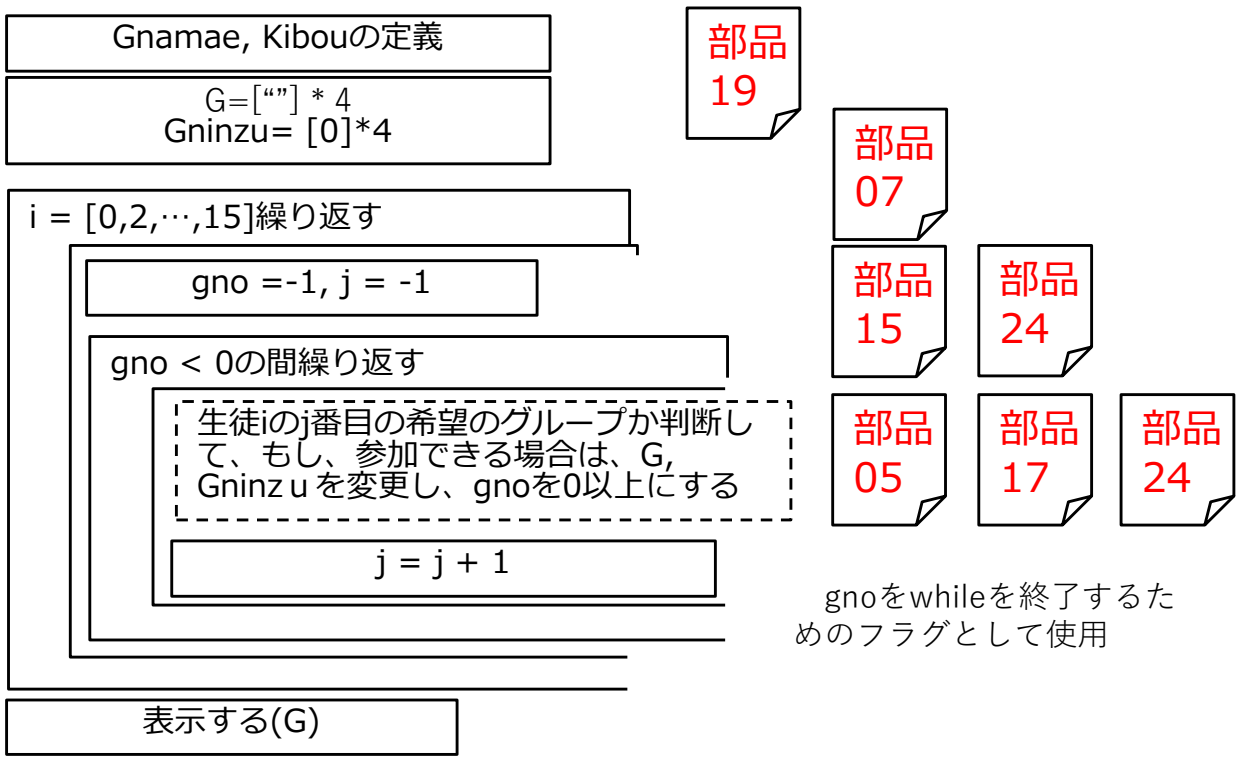
Kibou = [[0, 1, 2, 3],
[2, 1, 0, 3],
[0, 2, 1, 3],
・・・
[2, 1, 0, 3],
[0, 2, 1, 3]]

考え方

表示するための、各グループの生徒の氏名は、G=[“”,””,””,””]というリストの各要素に参加する生徒の氏名”A”, “B”などの追加する。また各グループの現在の人数を記録するためにGninzu= [0,0,0,0]というリストが管理する・

優先順番の並びで、 生徒ごとにどのグループに割り当てるか処理する。希望の上位のものから判断して、もし希望のグループがグループの人数の4名以下であれば、そのグループに参加できる。4名以上の場合は、その生徒の次の希望のグループに参加できるか判断する。

ヒント



パリティチェック(誤り検出/訂正)

パリティチェックは通信やメモリ上のデータの誤りを検出したり訂正する技術である。例えば左下図のような各7ビットのデータがあった場合、この場合データとパリティビット(P)の合計が偶数になるように、パリティビットを付加する。そして、テータをチェックするとき、例えばData2では、合計が奇数になっているので、どこかのビットに誤りかあることが検出できる。さらに列(縦方向)についてもパリティビットを付加すると、この例では列1に誤りがあることが検出でき、このことから、Data2の列1に誤りがあることが分かり、このビットを訂正すればよいことがわかる。

例えば、Dのように定義された、データとパリティビットがあった場合、誤りのあるビットの位置を表示するプログラムを作成する。なお、誤りが無い場合は、999を表示する。

	0	1	2	3	4	5	6	P	
Data0	0	1	1	0	1	1	1	1	←偶数
Data1	1	1	0	1	0	0	1	0	←偶数
Data2	1	1	1	1	0	1	0	0	←奇数
Data3	0	1	0	0	0	1	0	0	←偶数
Data4	0	0	1	1	1	0	0	1	←偶数
Data5	1	0	0	0	1	1	1	0	←偶数
Data6	0	1	0	1	0	1	0	1	←偶数
P	1	0	1	0	1	1	1		

```
D = [[0,1,1,0,1,1,1,1],
     [1,1,0,1,0,0,1,0],
     [1,1,1,1,0,1,0,0],
     [0,1,0,0,0,1,0,0],
     [0,0,1,1,1,0,0,1],
     [1,0,0,0,1,1,1,0],
     [0,1,0,1,0,1,0,1],
     [1,0,1,0,1,1,1,1]]
```

出力例

2 1　←　誤りが検出された場合

999　←　誤りがなかった場合

考え方

まず、パリティチェックの基本的な考え方として、上手のような7x7のデータでは、誤りが発生しても1ビットであるという仮定があり、このため誤りの箇所は無いか1個であると考える。　行(横)の誤り位置の情報をc, 列(縦)の誤り位置をrとして、cには初め999を入れて、①で誤りが見つかれば内容を変更する→999のままだと、②③で誤り無しと判断できる

① 行のData0からData 6 まで各行の合計(check_sum)が偶数になっているか判断し、そうでなければ、その行の番号をcに入れる。

②①で誤りが見つかった場合、列0～列6までの各工芸が偶数になっているか判断して、誤りの列を、rに入れる。

③①で誤りが見つかった場合は、cとrを表示する、そうでなければ999を表示する。

ヒント

D = 定義する

c = 999
r = 0
check_sum = 0

①行の誤り位置を検出する処理

②①で誤りが三つからなかった場合に、
列のの誤り位置を検出する処理

②①で誤りが三つからなかった場合は
999、誤りがあった場合はc,rを表示する
する処理

部品
11

部品
24

部品
17

部品
04

感染症の計算(微分計算:SIRモデル)

人口1000万人の都市において、初めに10人が感染していて、1000万人が未感染の状態で一日1人ひとりから1.2人に感染し、回復まで2日かかるような感染症があった場合に、60日間に、感染していない人、感染している人、回復した人がどのように変化するか表示するプログラムを作成する。

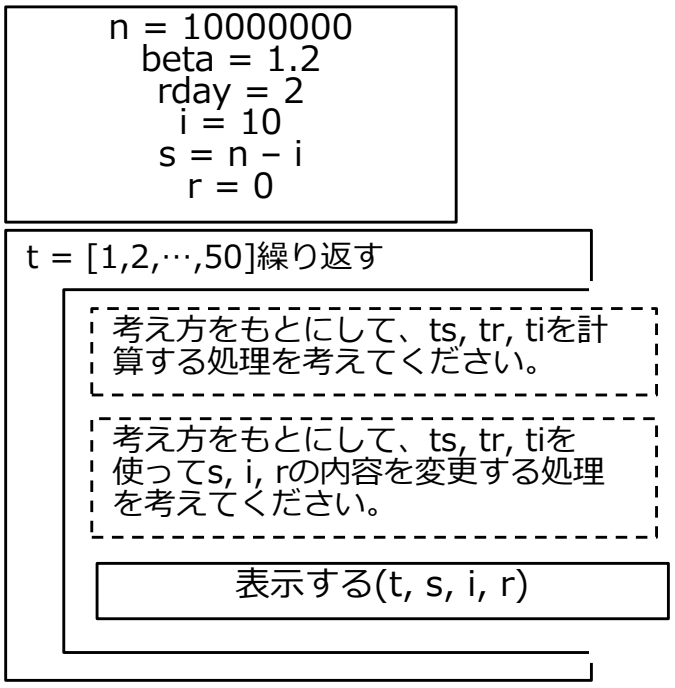
出力例 (日数, 感染していない人、感染した人、回復した人の順番で出力)の整数のみ表示
1 9999978 16 5
2 9999957 28 13
3 9999922 49 27
　～
58 852949 0 9147050
59 852949 0 9147050
60 852949 0 9147050

考え方

$$f(t+1) = f(t) + \Delta f(t)$$

感染していない人をs、感染した人i、回復した人をrとした場合、t+1日目の人数は、t日目の各s、i、rに対して各人数の変化分を足したものになる。プログラムとしては、各日の変化量の計算方法さえわかれば、単純に繰り返すだけの単純なものとなる。この問題で元の値の人口をn, 一人当たり感染人数をbeta、回復日数をrdayにした場合、各日の変化量をts、ti、trは、以下のようになる。
〇1日の回復者変化: tr = i * (1/ rday) ・・・(1/day)は一日に回復する人の割合
〇1日の感染していない人の変化 ts = i * (beta/n) * s ・・・新規に感染する人の人数が、感染していない人の変化である。betaはn人の時感染者に数なのでなので、感染者が増えてくると接触する人数自体が減ってくるため、感染する人数も変化すると考える。そのため(beta/n) は、未感染者が1人だった場合の感染人数の割合を示し、それに現在の未感染者数を掛けている。
〇1日の感染者数の変化 ti = ts – ti ・・・新規に感染した人から回復した人を秘したもの。
上記の変化量がわかったので、例えば、その日の回復した人の人数は
　r = r + tr となる。

ヒント



部品07

s, i, rは小数を含む数値になるため、
int(s)
として、整数部のみ表示している。

ファーストフードの待ち時間(待ち行列)

ファーストフードの窓口で、窓口でオーダーしてから品物が出来上がるまでの時間(待ち時間)をシミュレートするプログラムを作成する。条件としては、オーダーしてから、品物が出来上がるまでは、4分間調理にかかり、調理の間は次のオーダーをうけることが出来ない。そして客は1～10分間隔なランダムな時間に来店することで、100人の客が来店した時の平均待ち時間をシミュレートする。なお、出力は下記のように客の番号、来店時刻(分), オーダー開始時刻(分)、出来上がり時刻(分)、待ち時間(分)(時刻は開始からの経過時間を分で表す)を表示して、最後に平均待ち時間を表示する。

出力例(シミュレーションなので値は毎回違います)
0 5 5 9 4
1 8 9 13 5
2 14 14 18 4
～
98 549 549 553 4
99 556 556 560 4
5.54　←平均待ち時間

考え方

①来店時刻については、初めの客が、1～10分のランダムな時にきて、次の客はそれから、さらに1～10分のランダムな時にくることが繰り返される。各客の来店時刻についてa_timeとすると、ある客の来店時刻がa_timeとすると、次の客の来店時刻は、次のようにプログラミングできる。

a_time = atime + random.randint(1, 10)

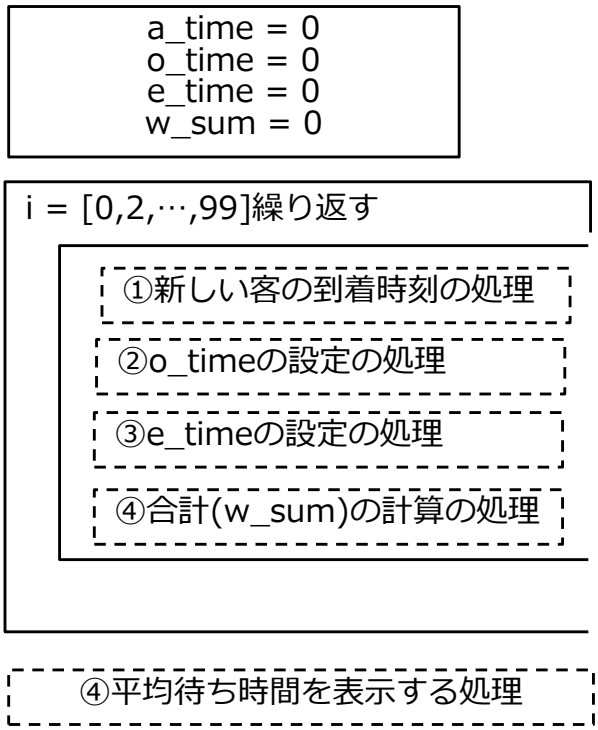
②オーダーと出来上がりの時刻に関しては、もし、前の客の出来上がり時刻がe_timeとすると。

e_time < a_timeの場合、すぐにオーダーできて、オーダー開始時刻をo_timeとすると、o_time = a_timeとなる。そうでなければ、前の客の出来上がり後にオーだできるので、o_time = e_timeになる。

③各客の出来上がり時間は o_time + 4となり、次の客から見ると、この値が②で使用するe_timeになる。各客の待ち時間は e_time – a_timeとなり、この待ち時間の合計(w_sum)を計算する。

④最後に、合計(w_sum)を100で割ったものが平気待ち時間になる。

ヒント



ロボット操作(1)

下左図のような中でロボットをスタート地点(S)からゴール地点(G)まで自動的に動かすプログラムを作成する。ロボットは初め(0,0)の位置にいて上を向いた状態に置く、そして(7,7)のゴールまで動かすが、ロボットの操作としては、　前に進む: F, 左を向く:L , 右を向く:Rの操作ができ、ゴールまでこのFLRの一連の操作の内容を文字列として出力する。

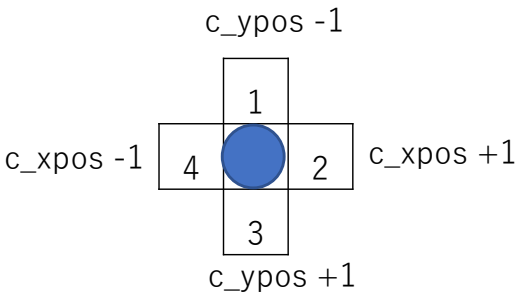
なお、道は一筆書きでかけるような一本道で、データとしては、二次元リストの形で、道は1, 通れない場所は0であらわしていて、プログラムはDのデータをもとに、一連の操作を自動的に出力するものである。

下図の場合の、出力例: **RFRFFFLFFRFFFLFFLFFFFRFFRFFFFF**

Y位置 \ X位置	0	1	2	3	4	5	6	7
0	S	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	1	0	0	0	1	1	1
3	0	1	1	1	0	1	0	1
4	0	0	0	1	0	1	0	1
5	0	0	0	1	0	1	0	1
6	0	0	0	1	1	1	0	1
7	0	0	0	0	0	0	0	G

```
D = [[1,1,0,0,0,0,0,0],
     [0,1,0,0,0,0,0,0],
     [0,1,0,0,0,1,1,1],
     [0,1,1,1,0,1,0,1],
     [0,0,0,1,0,1,0,1],
     [0,0,0,1,0,1,0,1],
     [0,0,0,1,1,1,0,1],
     [0,0,0,0,0,0,0,1]]
```

考え方



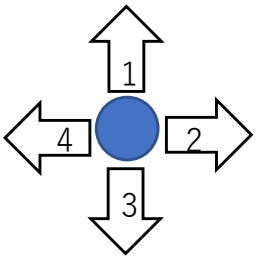
ロボットの現在の位置をc_xpos, c_yposとすると、次にロボットが移動できる位置は、c-ypos-1, c_ypos+1, c_xpos-1, 又はc_xposs+1のどれかになる。まさ、今回の課題では、一度通ったところには、戻らないので、一つ前の位置をb_xpos, b_yposとすると、この場所には移動しないとになる。

このことから、現在の位置、 c_xpos, c_yposと一つ前の位置b_xpos, b_yposと、全体のマップDを引き渡して、どちらの方向が進む方向であるか戻す関数をchek_wayを作成して、図のような進める位置(1～4)を戻り値とする。

次

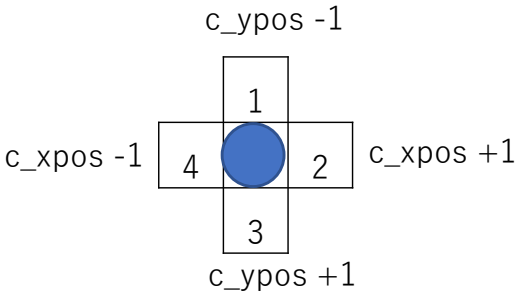
ロボット操作(2)

考え方



ロボットの現在の向きを右図のように1～4の状態で表すこととする。

また、現在の向きと、次の位置との関係を下記に示す。例えば、向きと次の位置が同じ場合は、F(前に進む)の操作をする、また向きが1で次の位置が2の場合は、RF(右を向く、前に進む)の操作になる。



	向き			
次の位置	1	2	3	4
1	F	LF	RRF	RF
2	RF	F	LF	RRF
3	RRF	RF	F	LF
4	LF	RRF	RF	F

向きと次の位置の関係をさらに見ていくと、その差をとった場合次のようになる(負の数になるので、(向き-次の位置+4)の余りとしている)
この表を利用すると、下記の操作となる。
0 = F
1 = LF
2 =RRF or LLF
3 = RF

	向き			
次の位置	1	2	3	4
1	0	1	2	3
2	3	0	1	2
3	2	3	0	1
4	1	2	3	0

D = 定義する

位置情報の初期化

部品 15

部品 22

c_xpos, c_yposがゴールまで繰り返す

check_wayの呼び出し

部品 18

b_xpos, b_yposを変更する

ロボットに対する操作を決める。そして、新しいロボットの向きを変更する処理を考えてください。

部品 23

c_xpos, c_yposを次のロボット位置に変更する。

表示する(O)

関数:checkway(
c_xpos, c_ypos,
b_xpos, b_ypos,
D)

考え方をもとにして、1～4を戻り値とする関数を考えてください。

戻る: v

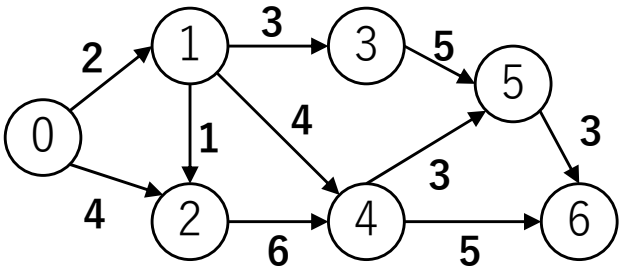
部品 18

部品 17

部品 22

最短経路(グラフ)(1)

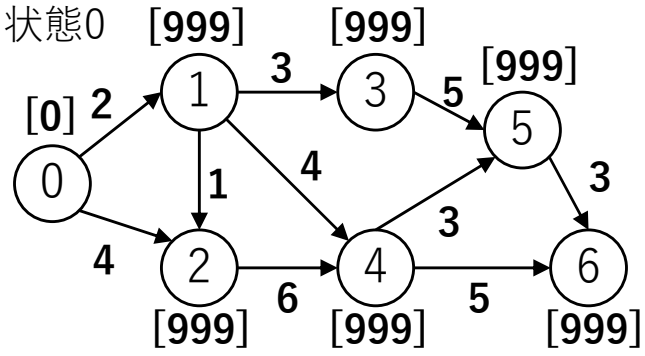
下図では⓪～⑥までの地点は、図のように移動が可能で、個々の道の距離を示している。例えば地点⓪～①までの距離は2である。**下図の場合に、各地点に最短でたどり着く距離を表示するプログラムを作成する。**なお、図自体の地点と距離の関係を表したDのようなデータで表現されていて、出力は、⓪～⑥までの採点経路で [0, 2, 4, 5, 6, 9, 11]のように表示される。




```
D =[[0, 1, 2],
    [0, 2, 4],
    [1, 2, 1],
    [1, 3, 3],
    [1, 4, 4],
    [2, 4, 6],
    [3, 5, 5],
    [4, 5, 3],
    [4, 6, 5],
    [5, 6, 3]]
```

[地点元, 地点先,距離]

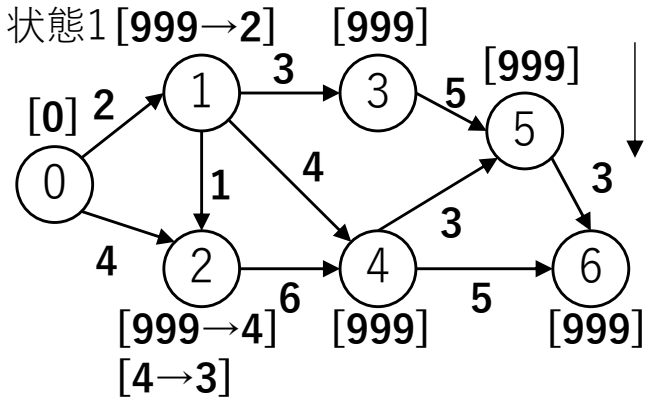
考え方(1)



この課題のような、地点(頂点/ノード)を道(辺/エッジ)を使用して、物の結びつきを表したものをグラフと呼ぶ。世の中のいろいろな事象は、このような物の結びつきで表現することができ。このグラフをもとに、いろいろな問題のアルゴリズムで解決していくことが多い。

特にグラフ問題を解くアルゴリズムはいろいろな考えたがあり、この課題では、その一つである、ベルマン・フォード法を利用してみる。

まず、この方法では、グラフをDのように、二つの地点(頂点/ノード)の距離で表現するように道(辺/エッジ)で表す。そして、各地点の最短距離を順番に求めていくことになる。
まず、上図:状態0のように、開始点は距離が0なので、最短距離を0に設定する。そして、他の地点については、初期状態では、非常に大きな数を設定しておく。



[地点元,　地点先,　距離]
[　0,　　1,　　2], ← ここ1番目処理
[　0,　　2,　　4], ← ここ2番目処理
[　1,　　2,　　1], ← ここ3番目処理
～

次に、Dのデータを見て、地点先の最短距離を更新していく、この場合、地点先に注目して、初めのデータの地点1のデータと地点0からの距離をもとにして比較する
地点先の距離　　地点元の距離と距離
[999]　　と　　[0] + 2
で、[999]の方が大きいので、地点①の距離を[2]に変更する。

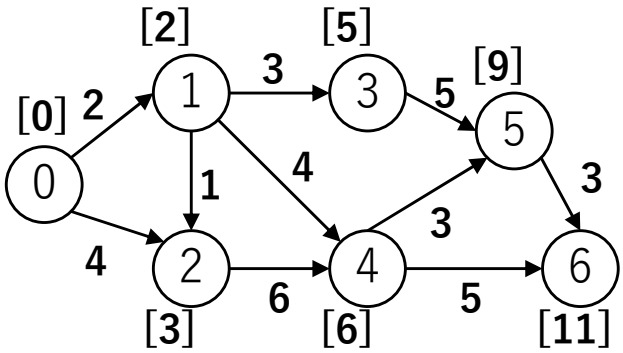
2番目の処理では、同様な処理をして、地点②の距離が[4]になる。
3番目の処理では 現在の地点③の距離[4] と地点①の距離[2]+1を比較して、地点③の距離を[4]から[3]に変更する。

次

最短経路(グラフ)(2)

考え方(2)

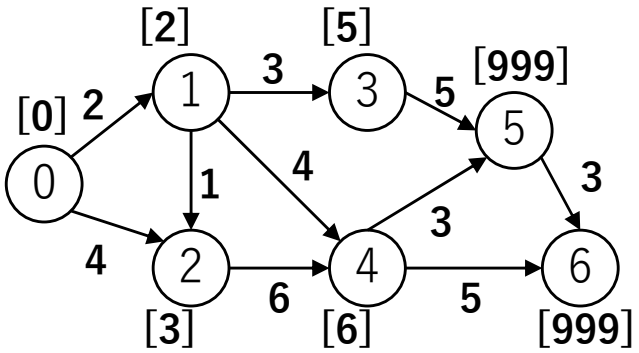
データが最適に並んでいる場合




```
D =[[0, 1, 2],
    [0, 2, 4],
    [1, 2, 1],
    [1, 3, 3],
    [1, 4, 4],
    [2, 4, 6],
    [3, 5, 5],
    [4, 5, 3],
    [4, 6, 5],
    [5, 6, 3]]
```

実行結果[0, 2, 4, 5, 6, 9, 11]

データがランダムに並んでいる場合




```
D =[[0, 1, 2],
    [4, 6, 5],
    [4, 5, 3],
    [0, 2, 4],
    [3, 5, 5],
    [1, 2, 1],
    [1, 3, 3],
    [1, 4, 4],
    [2, 4, 6],
    [5, 6, 3]]
```

実行結果[[0, 2, 3, 5, 6, 999, 999]]

このアルゴリズムを使用した場合、地点と道のデータが地点①～⑥まで順番に計算されるように適切に並んでいた場合は、1回の処理で正しく、最短距離が求まる。
ただし、ランダムに並んだいた場合は、一つ前の地点の最短距離が確定しない場合、必ずしも1回の処理で正しく、すべての地点の最短距離が求められない。
このため、1回の処理で最低1地点の最短距離が決まるとして、地点の数だけ処理を繰り返すと、すべての地点の最短距離が求められことになる。

次

最短経路(グラフ)(3)

考え方(3)

D = 定義する

Dの各要素は
[地点元, 地点先,距離]
D[?][1] D[?][2] D[?][3]で使用する

n = 7

n は地点の数

Cost = [999] * 7

Costは各地点の最短距離を格納するリスト。
すべての値を999で初期化しておく。
Cost = [999, 999, 999, ・・・,999]

Cost[0] = 0

地点①の最短距離として0を設定

i = [0,2,…,n-1]繰り返す

j= [0,2,…,Dの大きさ-1]繰り返す

x = 地点元
y = 地点先
z = x-yの距離　とすると
地点先の距離Cost[y]が地点元の距離[x]+zより大きければ、
地点先の距離Cost[y]=地点元の距離[x]+zに
する処理を考えてください。
[補足]

部品
11

部品
17

部品
04

表示する(Cost)

補足

前のページで説明したように、データの並び方によっては、かなり早い時に、すべての地点の最短距離が求まり、それ以降は無駄な処理を繰り返すことになる。
実際の効率的なアルゴリズムがでは、この無駄を避けるため、上図の補足の位置で、最短距離が置き換わったら、「j= [0,2,…,Dの大きさ-1]繰り返す」を抜け出す処理をしている。

部品
15

部品
24