

#Benedicts'

As minsteps is 5 steps, process from 7 steps to 5 step first for number of possible pathway, then brute force.

modified time_subpath for each value after ':' for the time, in both reverse and normal sequence

```
time_subpath = {'ab':6,'ba':6,'ra':3,'ar':3,'ac':4,'ca':4,'br':6,'rb':6,'cr':1,'rc':1,'bc':4,'cb':4}
```

```
import itertools
```

```
import numpy as np
```

```
import pprint
```

```
total_path = {}
```

```
weight_subpath = {}
```

```
map = ['r', 'a', 'b', 'c']
```

```
set_7 = () # this is set_7
```

```
set_6 = ()
```

```
set_5 = ()
```

```
path = {0: ['a', 'a', 'a', 'a', 'a', 'a', 'a']}
```

```
total_time = 0
```

```
shortest_time = 999999
```

```
shortest_pathway = {}
```

```
all_shortest_pathway_s = {}
```

```
all_shortest_pathway_s_count = 0
```

```
for p in itertools.product(map, repeat=7):
```

```
    set_7 += p
```

```
set_7 = np.array_split(set_7, 16384)
```

Possible Pathways filter for 7 step

```
a = 0
```

```
n = 0
```

```
end = 16384
```

```
while n < end:
```

```
    if set_7[n][0] != 'r' or set_7[n][-1] != 'r':
```

```
        del set_7[n]
```

```
        n -= 1
```

```
        end -= 1
```

```
    n += 1
```

```
n = 0
```

```
n1 = 0
```

```
end = len(set_7)
```

```
tamp = 0
```

```
while n < end:
```

```
    n1 = 0
```

```
    while n1 < 6:
```

```
        n2 = n1 + 1
```

```
        if set_7[n][n1] == set_7[n][n2] and set_7[n][n1] == 'r':
```

```
            del set_7[n]
```

```
            n -= 1
```

```
            end -= 1
```

```
            break
```

```
        n1 += 1
```

```
    n += 1
```

```
n = 0
```

```
n1 = 0
```

```
oper_a = False
```

```
end = len(set_7)
```

```
while n < end:
```

```
    n1 = 0
```

```
    oper_a = False
```

```
    while n1 < 6:
```

```
        if set_7[n][n1] == 'a':
```

```

oper_a = True
while True:
    n1 += 1
    if n1 == 7:
        break
    if oper_a == True and set_7[n][n1] == 'a':
        del set_7[n]
        n -= 1
        end -= 1
        break
    if oper_a == True:
        break
    else:
        n1 += 1
n += 1

```

```

n = 0
n1 = 0
oper_b = False
end = len(set_7)
while n < end:
    n1 = 0
    oper_b = False
    while n1 < 6:
        if set_7[n][n1] == 'b':
            oper_b = True
            while True:
                n1 += 1
                if n1 == 7:
                    break
            if oper_b == True and set_7[n][n1] == 'b':
                del set_7[n]
                n -= 1
                end -= 1
                break
        if oper_b == True:
            break
        else:
            n1 += 1
    n += 1

```

```

n = 0
n1 = 0
oper_c = False
end = len(set_7)
while n < end:
    n1 = 0
    oper_c = False
    while n1 < 6:
        if set_7[n][n1] == 'c':
            oper_c = True
            while True:
                n1 += 1
                if n1 == 7:
                    break
            if oper_c == True and set_7[n][n1] == 'c':
                del set_7[n]
                n -= 1
                end -= 1
                break
        if oper_c == True:
            break
        else:
            n1 += 1
    n += 1

```

```
    else:
        n1 += 1
    n += 1
```

#below is 6 steps

```
for p in itertools.product(map, repeat=6):
    set_6 += p
```

```
set_6 = np.array_split(set_6, 4096)
```

Possible Pathways filter for 6 step

```
a = 0
n = 0
end = 4096
while n < end:
    if set_6[n][0] != 'r' or set_6[n][-1] != 'r':
        del set_6[n]
        n -= 1
        end -= 1
    n += 1
```

```
n = 0
n1 = 0
end = len(set_6)
tamp = 0
while n < end:
    n1 = 0
    while n1 < 5:
        n2 = n1 + 1
        if set_6[n][n1] == set_6[n][n2] and set_6[n][n1] == 'r':
            del set_6[n]
            n -= 1
            end -= 1
            break
        n1 += 1
    n += 1
```

```
n = 0
n1 = 0
oper_a = False
end = len(set_6)
while n < end:
    n1 = 0
    oper_a = False
    while n1 < 5:
        if set_6[n][n1] == 'a':
            oper_a = True
            while True:
                n1 += 1
                if n1 == 6:
                    break
            if oper_a == True and set_6[n][n1] == 'a':
                del set_6[n]
                n -= 1
                end -= 1
                break
        if oper_a == True:
            break
    else:
        n1 += 1
    n += 1
```

```

n = 0
n1 = 0
oper_b = False
end = len(set_6)
while n < end:
    n1 = 0
    oper_b = False
    while n1 < 5:
        if set_6[n][n1] == 'b':
            oper_b = True
            while True:
                n1 += 1
                if n1 == 6:
                    break
            if oper_b == True and set_6[n][n1] == 'b':
                del set_6[n]
                n -= 1
                end -= 1
                break
        if oper_b == True:
            break
        else:
            n1 += 1
    n += 1

```

```

n = 0
n1 = 0
oper_c = False
end = len(set_6)
while n < end:
    n1 = 0
    oper_c = False
    while n1 < 5:
        if set_6[n][n1] == 'c':
            oper_c = True
            while True:
                n1 += 1
                if n1 == 6:
                    break
            if oper_c == True and set_6[n][n1] == 'c':
                del set_6[n]
                n -= 1
                end -= 1
                break
        if oper_c == True:
            break
        else:
            n1 += 1
    n += 1

```

#below is 5 steps

```

for p in itertools.product(map, repeat=5):
    set_5 += p

```

```

set_5 = np.array_split(set_5, 1024)

```

Possible Pathways filter for 5 step different from 5 and 6 no r check after r repetition but set[n][1,2,3] can't be r

```

a = 0
n = 0
end = 1024
while n < end:

```

```
if set_5[n][0] != 'r' or set_5[n][-1] != 'r':
    del set_5[n]
    n -= 1
end -= 1
n += 1
```

```
a = 0
n = 0
```

```
while n < len(set_5):
    if set_5[n][1] == 'r' or set_5[n][2] == 'r' or set_5[n][3] == 'r':
        del set_5[n]
        n -= 1
    end -= 1
n += 1
```

```
n = 0
n1 = 0
```

```
oper_a = False
end = len(set_5)
```

```
while n < end:
    n1 = 0
    oper_a = False
    while n1 < 4:
        if set_5[n][n1] == 'a':
            oper_a = True
            while True:
                n1 += 1
                if n1 == 5:
                    break
            if oper_a == True and set_5[n][n1] == 'a':
                del set_5[n]
                n -= 1
                end -= 1
                break
        if oper_a == True:
            break
        else:
            n1 += 1
    n += 1
```

```
n = 0
n1 = 0
```

```
oper_b = False
end = len(set_5)
```

```
while n < end:
    n1 = 0
    oper_b = False
    while n1 < 4:
        if set_5[n][n1] == 'b':
            oper_b = True
            while True:
                n1 += 1
                if n1 == 5:
                    break
            if oper_b == True and set_5[n][n1] == 'b':
                del set_5[n]
                n -= 1
                end -= 1
                break
        if oper_b == True:
            break
        else:
```

```
    n1 += 1
n += 1
```

```
n = 0
n1 = 0
oper_c = False
end = len(set_5)
while n < end:
    n1 = 0
    oper_c = False
    while n1 < 4:
        if set_5[n][n1] == 'c':
            oper_c = True
            while True:
                n1 += 1
                if n1 == 5:
                    break
                if oper_c == True and set_5[n][n1] == 'c':
                    del set_5[n]
                    n -= 1
                    end -= 1
                    break
            if oper_c == True:
                break
            else:
                n1 += 1
    n += 1
```

```
total_path = len(set_5)+len(set_6)+len(set_7)
all_set = set_5 + set_6 + set_7
```

```
print('\nTotal possible pathway is {}'.format(total_path))
print('\nAll pathway display out -> {}'.format(all_set))
```

#double loop for brute force to found out 1st shortest pathway

```
n=0
n1=0
between=""
total_time = 0
shortest_time = 999999
shortest_pathway = []
while n < total_path:
    total_time = 0
    n1 = 0
    while n1 < len(all_set[n])-1:
        between = str(all_set[n][n1]+all_set[n][n1+1])
        total_time = time_subpath[between] + total_time
        if(n1 == len(all_set[n])-2):
            if shortest_time > total_time:
                shortest_time = total_time
                shortest_pathway = all_set[n]
        n1 += 1
    n += 1
```

```
print('\nShortest time is {}'.format(shortest_time))
print('\nAll possible pathways is listed below:')
```

#loop for all possible outcome and print it out automatically

```
n=0
n1=0
between=""
total_time = 0
while n < total_path:
    total_time = 0
    n1 = 0
```

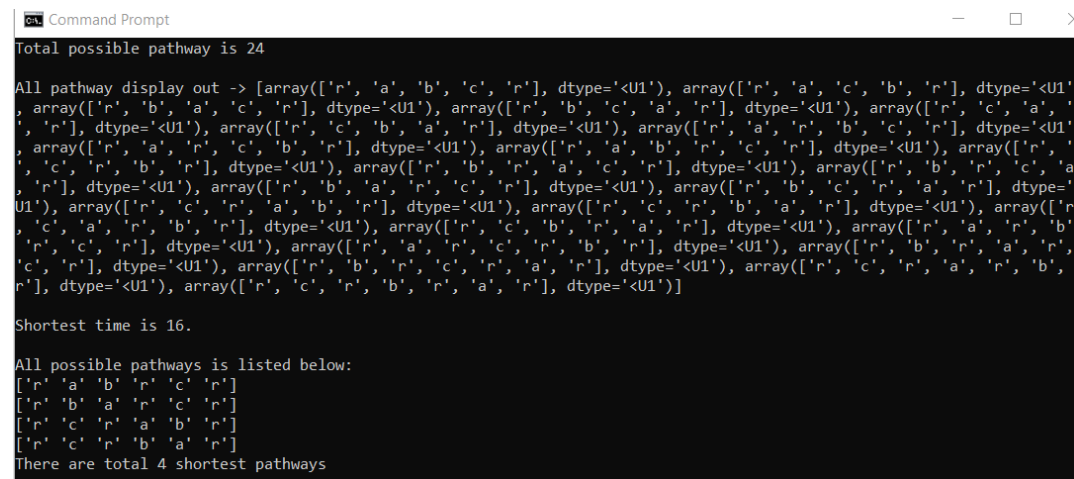
```
while n1 < len(all_set[n])-1:
    between = str(all_set[n][n1]+all_set[n][n1+1])
    total_time = time_subpath[between] + total_time
    if(n1 == len(all_set[n])-2):
        if shortest_time == total_time:
            all_shortest_pathway_s = all_set[n]
            print(all_shortest_pathway_s)
            all_shortest_pathway_s_count +=1
        n1 += 1
    n += 1
print('There are total {} shortest pathways'.format(all_shortest_pathway_s_count))
```

1a) There are 4 most efficient pathway: RABRCR, RBARCR, RCRABR and RCRBAR.

With time =16

I know as I made a python program for this question (wrote in pycharm IDE). See the file I uploaded. I import numpy and itertools (default have) only into the program.

Below picture is the output of the program:



```
Command Prompt
Total possible pathway is 24

All pathway display out -> [array(['r', 'a', 'b', 'c', 'r'], dtype='<U1'), array(['r', 'a', 'c', 'b', 'r'], dtype='<U1'),
array(['r', 'b', 'a', 'c', 'r'], dtype='<U1'), array(['r', 'b', 'c', 'a', 'r'], dtype='<U1'), array(['r', 'c', 'a', 'b',
', 'r'], dtype='<U1'), array(['r', 'c', 'b', 'a', 'r'], dtype='<U1'), array(['r', 'a', 'r', 'b', 'c', 'r'], dtype='<U1'),
array(['r', 'a', 'r', 'c', 'b', 'r'], dtype='<U1'), array(['r', 'a', 'b', 'r', 'c', 'r'], dtype='<U1'), array(['r', 'a',
', 'c', 'r', 'b', 'r'], dtype='<U1'), array(['r', 'b', 'r', 'a', 'c', 'r'], dtype='<U1'), array(['r', 'b', 'r', 'c', 'a',
', 'r'], dtype='<U1'), array(['r', 'b', 'a', 'r', 'c', 'r'], dtype='<U1'), array(['r', 'b', 'c', 'r', 'a', 'r'], dtype='<
U1'), array(['r', 'c', 'r', 'a', 'b', 'r'], dtype='<U1'), array(['r', 'c', 'r', 'b', 'a', 'r'], dtype='<U1'), array(['r
', 'c', 'a', 'r', 'b', 'r'], dtype='<U1'), array(['r', 'c', 'b', 'r', 'a', 'r'], dtype='<U1'), array(['r', 'a', 'r', 'b',
', 'r'], dtype='<U1'), array(['r', 'a', 'r', 'c', 'r', 'b', 'r'], dtype='<U1'), array(['r', 'b', 'r', 'a', 'r', 'c',
', 'r'], dtype='<U1'), array(['r', 'b', 'r', 'c', 'r', 'a', 'r'], dtype='<U1'), array(['r', 'c', 'r', 'a', 'r', 'b',
', 'r'], dtype='<U1'), array(['r', 'c', 'r', 'b', 'r', 'a', 'r'], dtype='<U1')]

Shortest time is 16.

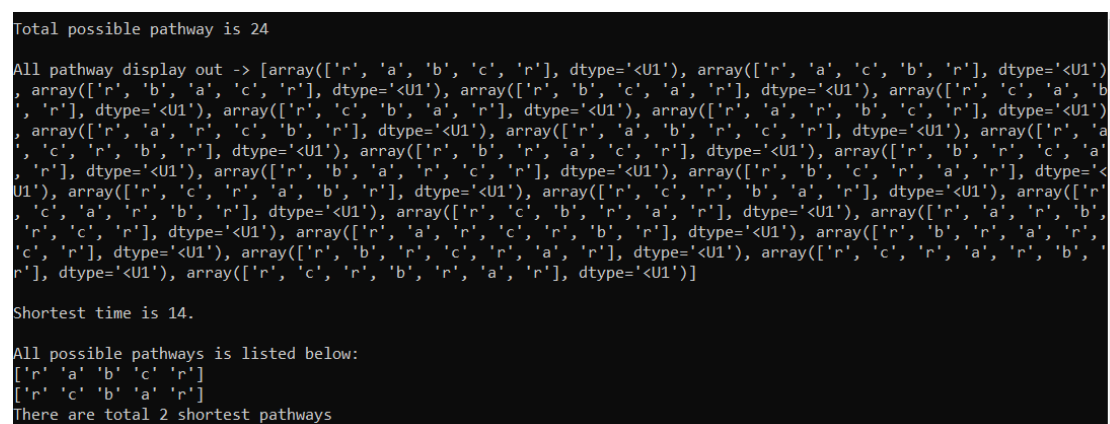
All possible pathways is listed below:
['r', 'a', 'b', 'r', 'c', 'r']
['r', 'b', 'a', 'r', 'c', 'r']
['r', 'c', 'r', 'a', 'b', 'r']
['r', 'c', 'r', 'b', 'a', 'r']
There are total 4 shortest pathways
```

I first generate all pathways for 5(min step), 6, 7(max step) step pathways by itertools.product, then use custom loop to filter out impossible pathways. Though I think this is very inefficient way.

Then I brute force all possible pathways to find all possible shortest pathways, with shortest time.

1b) I made a program to calculate out shortest pathways.

2a) update the time value in the program mentioned, new output:



```
Command Prompt
Total possible pathway is 24

All pathway display out -> [array(['r', 'a', 'b', 'c', 'r'], dtype='<U1'), array(['r', 'a', 'c', 'b', 'r'], dtype='<U1'),
array(['r', 'b', 'a', 'c', 'r'], dtype='<U1'), array(['r', 'b', 'c', 'a', 'r'], dtype='<U1'), array(['r', 'c', 'a', 'b',
', 'r'], dtype='<U1'), array(['r', 'c', 'b', 'a', 'r'], dtype='<U1'), array(['r', 'a', 'r', 'b', 'c', 'r'], dtype='<U1'),
array(['r', 'a', 'r', 'c', 'b', 'r'], dtype='<U1'), array(['r', 'a', 'b', 'r', 'c', 'r'], dtype='<U1'), array(['r', 'a',
', 'c', 'r', 'b', 'r'], dtype='<U1'), array(['r', 'b', 'r', 'a', 'c', 'r'], dtype='<U1'), array(['r', 'b', 'r', 'c', 'a',
', 'r'], dtype='<U1'), array(['r', 'b', 'a', 'r', 'c', 'r'], dtype='<U1'), array(['r', 'b', 'c', 'r', 'a', 'r'], dtype='<
U1'), array(['r', 'c', 'r', 'a', 'b', 'r'], dtype='<U1'), array(['r', 'c', 'r', 'b', 'a', 'r'], dtype='<U1'), array(['r
', 'c', 'a', 'r', 'b', 'r'], dtype='<U1'), array(['r', 'c', 'b', 'r', 'a', 'r'], dtype='<U1'), array(['r', 'a', 'r', 'b',
', 'r'], dtype='<U1'), array(['r', 'a', 'r', 'c', 'r', 'b', 'r'], dtype='<U1'), array(['r', 'b', 'r', 'a', 'r', 'c',
', 'r'], dtype='<U1'), array(['r', 'b', 'r', 'c', 'r', 'a', 'r'], dtype='<U1'), array(['r', 'c', 'r', 'a', 'r', 'b',
', 'r'], dtype='<U1'), array(['r', 'c', 'r', 'b', 'r', 'a', 'r'], dtype='<U1')]

Shortest time is 14.

All possible pathways is listed below:
['r', 'a', 'b', 'c', 'r']
['r', 'c', 'b', 'a', 'r']
There are total 2 shortest pathways
```

Thus, new shortest pathways = RABCR and RCBAR

2)b) I updated the new time in the program and let it recompute new shortest pathways.

3)a)

Rewrite my program dictionary "time_subpath" to "time_subpath = {'ab':x,'ba':x,'ra':w,'ar':w,'ac':y,'ca':y,'br':u,'rb':u,'cr':v,'rc':v,'bc':z,'cb':z}" at line 5

According to new time, then calculate new shortest pathway again. And remind he/she to run with python 3 with numpy imported, this programe will work if total time =< 999999 they can change the limit at "shortest_time = 999999" at line 368.