Alternative solution to np.inf in the first greedy approach

Emanuele Carelli
Github: https://github.com/imEmaa

Let a = distance_matrix[city], b = visited, c our ‘mask’

a	b	c
T	T	F
T	F	T
F	T	F
F	F	F

This is the desired behavior

Easily we understand that c = A & (not B)

a	b	x=not(b)	c=a&n(b)
T	T	F	F
T	F	T	T
F	T	F	F
F	F	T	F

Actually, if A=F means B = T, so last entry is not possible
The mask simply is not(b)

But what does mask actually do?

We use mask to filter distance_matrix[city]: we want to take a city not visited with distance > 0 and leave a city visited

Actually, we don’t neet to perform a & not(b): mask can be defined with only not(b), because when we visit a new city we put a True value in visited[city] (a=F, b=F not possible in our implementation)

So, mask is simply np.logical_not(visited)

How we find the minimum value?

When we compute dist_matrix[city][mask] we only check in the position marked as True in the mask: so, calculating the argmin gives us the index inside the masked array.

To get the cardinality of that true, we simply add 1 (if argmin = 0 we refer to the first True value)

Let’s use an example and suppose this is our final result:

- a = np.array([0, 2, 4, 1, 5]), not(b) = False, True, False, True, True

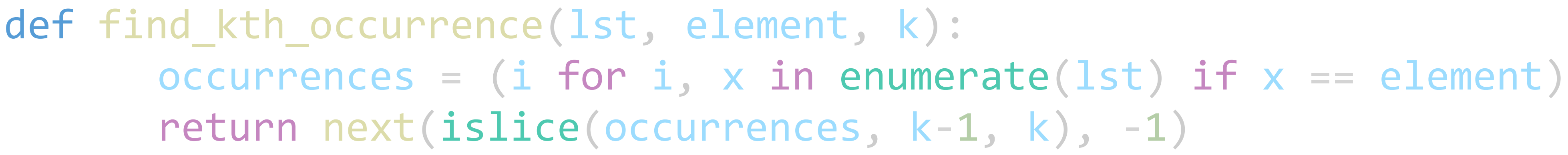
The np.argmin(a[mask]) returns us 1

- we are looking for the 1+1 True value corrispondance in the mask

The second True value has index = 3. a[3] = 1, which is the minimum value

Find k-occurrence

We define a function

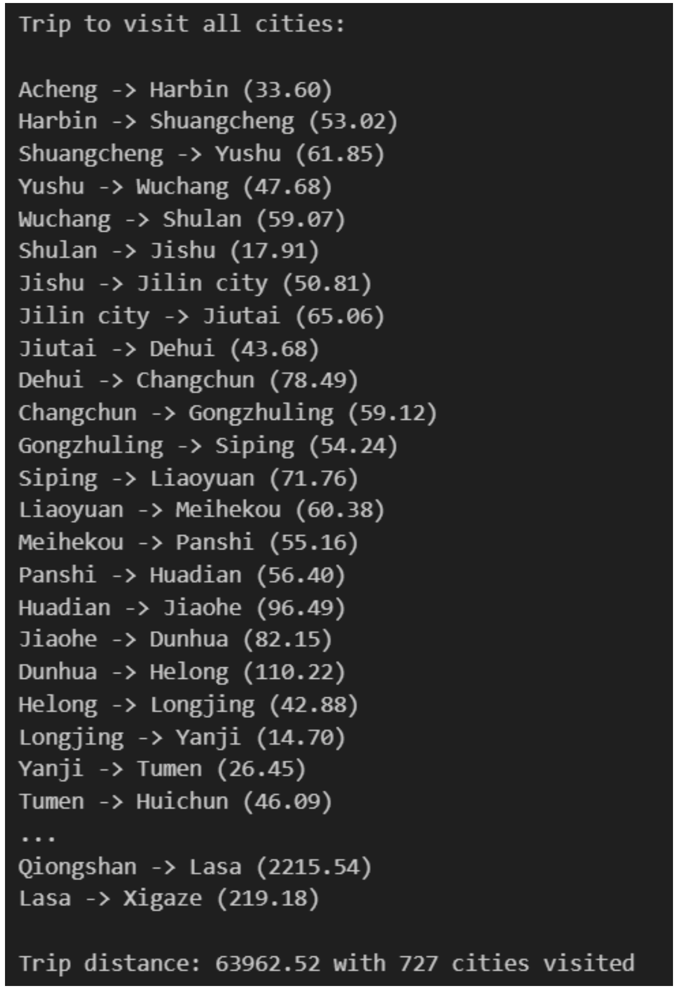


```
def find_kth_occurrence(lst, element, k):
    occurrences = (i for i, x in enumerate(lst) if x == element)
    return next(islice(occurrences, k-1, k), -1)
```

to find the k-occurrence of true value in the vector, k=argmin()+1

So we finally get our index of the min in the dist_matrix, computing a not operation and using a find_kth_occurrence function

Bonus: China sample comparison

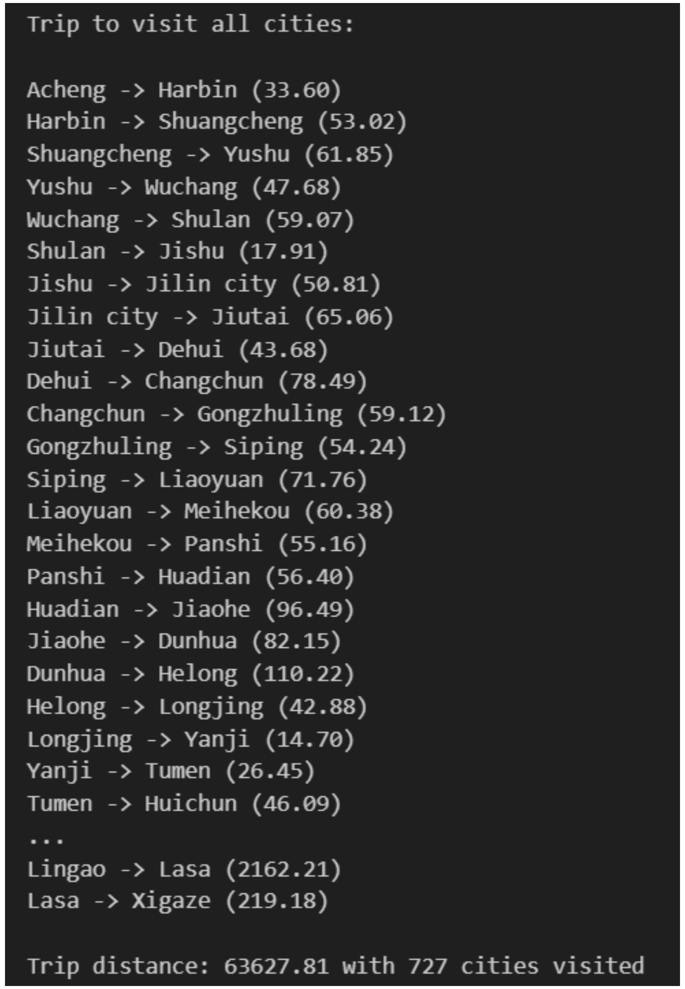


```
Trip to visit all cities:

Acheng -> Harbin (33.60)
Harbin -> Shuangcheng (53.02)
Shuangcheng -> Yushu (61.85)
Yushu -> Wuchang (47.68)
Wuchang -> Shulan (59.07)
Shulan -> Jishu (17.91)
Jishu -> Jilin city (50.81)
Jilin city -> Jiutai (65.06)
Jiutai -> Dehui (43.68)
Dehui -> Changchun (78.49)
Changchun -> Gongzhuling (59.12)
Gongzhuling -> Siping (54.24)
Siping -> Liaoyuan (71.76)
Liaoyuan -> Meihekou (60.38)
Meihekou -> Panshi (55.16)
Panshi -> Huadian (56.40)
Huadian -> Jiaohe (96.49)
Jiaohe -> Dunhua (82.15)
Dunhua -> Helong (110.22)
Helong -> Longjing (42.88)
Longjing -> Yanji (14.70)
Yanji -> Tumen (26.45)
Tumen -> Huichun (46.09)
...
Qiongshan -> Lasa (2215.54)
Lasa -> Xigaze (219.18)

Trip distance: 63962.52 with 727 cities visited
```

Using np.inf



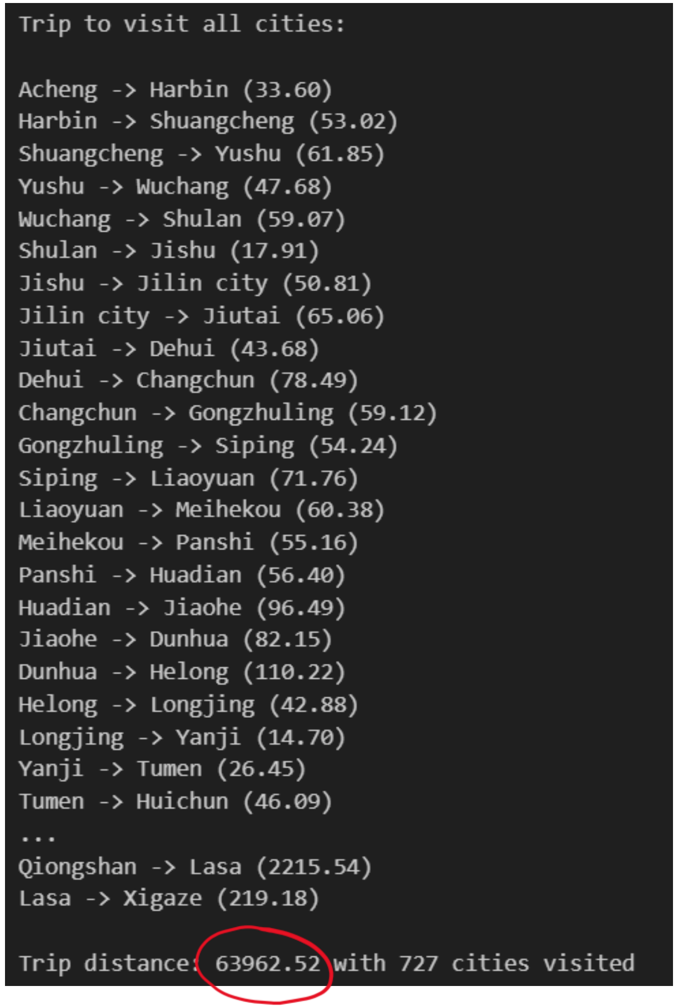
```
Trip to visit all cities:

Acheng -> Harbin (33.60)
Harbin -> Shuangcheng (53.02)
Shuangcheng -> Yushu (61.85)
Yushu -> Wuchang (47.68)
Wuchang -> Shulan (59.07)
Shulan -> Jishu (17.91)
Jishu -> Jilin city (50.81)
Jilin city -> Jiutai (65.06)
Jiutai -> Dehui (43.68)
Dehui -> Changchun (78.49)
Changchun -> Gongzhuling (59.12)
Gongzhuling -> Siping (54.24)
Siping -> Liaoyuan (71.76)
Liaoyuan -> Meihekou (60.38)
Meihekou -> Panshi (55.16)
Panshi -> Huadian (56.40)
Huadian -> Jiaohe (96.49)
Jiaohe -> Dunhua (82.15)
Dunhua -> Helong (110.22)
Helong -> Longjing (42.88)
Longjing -> Yanji (14.70)
Yanji -> Tumen (26.45)
Tumen -> Huichun (46.09)
...
Lingao -> Lasa (2162.21)
Lasa -> Xigaze (219.18)

Trip distance: 63627.81 with 727 cities visited
```

Using mask

Bonus: China sample comparison

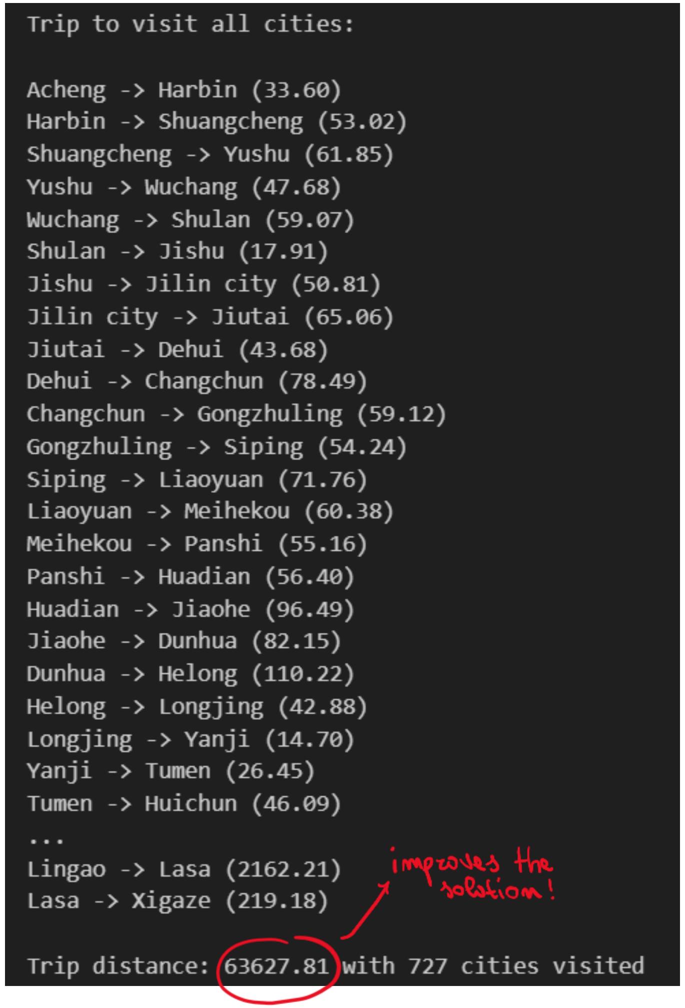


```
Trip to visit all cities:

Acheng -> Harbin (33.60)
Harbin -> Shuangcheng (53.02)
Shuangcheng -> Yushu (61.85)
Yushu -> Wuchang (47.68)
Wuchang -> Shulan (59.07)
Shulan -> Jishu (17.91)
Jishu -> Jilin city (50.81)
Jilin city -> Jiutai (65.06)
Jiutai -> Dehui (43.68)
Dehui -> Changchun (78.49)
Changchun -> Gongzhuling (59.12)
Gongzhuling -> Siping (54.24)
Siping -> Liaoyuan (71.76)
Liaoyuan -> Meihekou (60.38)
Meihekou -> Panshi (55.16)
Panshi -> Huadian (56.40)
Huadian -> Jiaohe (96.49)
Jiaohe -> Dunhua (82.15)
Dunhua -> Helong (110.22)
Helong -> Longjing (42.88)
Longjing -> Yanji (14.70)
Yanji -> Tumen (26.45)
Tumen -> Huichun (46.09)
...
Qiongshan -> Lasa (2215.54)
Lasa -> Xigaze (219.18)

Trip distance: 63962.52 with 727 cities visited
```

Using np.inf



```
Trip to visit all cities:

Acheng -> Harbin (33.60)
Harbin -> Shuangcheng (53.02)
Shuangcheng -> Yushu (61.85)
Yushu -> Wuchang (47.68)
Wuchang -> Shulan (59.07)
Shulan -> Jishu (17.91)
Jishu -> Jilin city (50.81)
Jilin city -> Jiutai (65.06)
Jiutai -> Dehui (43.68)
Dehui -> Changchun (78.49)
Changchun -> Gongzhuling (59.12)
Gongzhuling -> Siping (54.24)
Siping -> Liaoyuan (71.76)
Liaoyuan -> Meihekou (60.38)
Meihekou -> Panshi (55.16)
Panshi -> Huadian (56.40)
Huadian -> Jiaohe (96.49)
Jiaohe -> Dunhua (82.15)
Dunhua -> Helong (110.22)
Helong -> Longjing (42.88)
Longjing -> Yanji (14.70)
Yanji -> Tumen (26.45)
Tumen -> Huichun (46.09)
...
Lingao -> Lasa (2162.21)
Lasa -> Xigaze (219.18)

Trip distance: 63627.81 with 727 cities visited
```

improves the solution!

Using mask

Code

```
# Initialization



def find_kth_occurrence(lst, element, k):

    occurrences = (i for i, x in enumerate(lst) if x == element)

    return next(islice(occurrences, k-1, k), -1)

visited = np.full((cities.shape[0]), False)

city = 0

visited[city] = True

trip = []

trip.append(city)

print("Trip to visit all cities:\n")
```

```
# Cycle


while not np.all(visited):

    # visited = np.array(visited, dtype=bool)

    # support = np.logical_and(distance_matrix[city], np.logical_not(visited))

    support = np.logical_not(visited)

    k = np.argmin(distance_matrix[city][support])

    closest_city = find_kth_occurrence(support, 1, k+1)

    print(f"{names[city]} -> {names[closest_city]}
({distance_matrix[city][closest_city]:.2f})")

    city = closest_city

    visited[city] = True

    trip.append(city)

trip.append(0)
```