钻石金字塔问题

2017211305 班
2017211240 于海鑫

December 7, 2019

1 问题描述

1.1 基础问题

给定一个 n 行的三角形金字塔，你现在所在的位置为（0，0）。你的移动方式被限定为只能往左下方或者右下方走，即 $(x,y) \rightarrow (x+1,y) \mid (x,y+1)$。对于每一个点有一个价值 $value[i][j]$ 表示你走到这里能够获得多少金币。请你找到一条从第 0 层到第 $n-1$ 层的路线，使得能够获得最大数目的金币。

1.2 进阶版本（蒙图版）

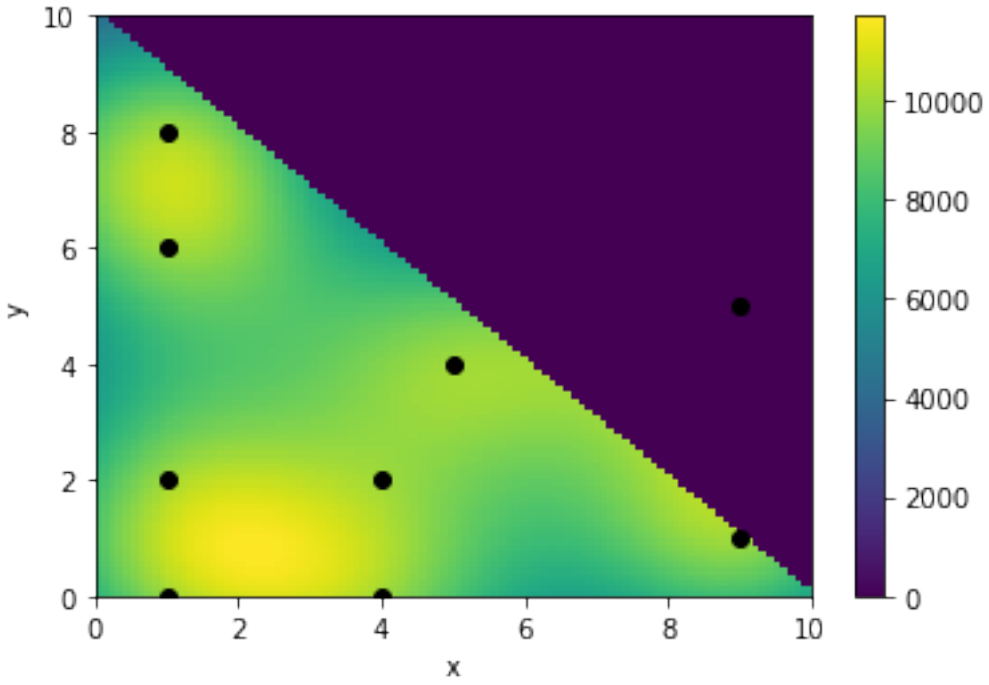
假设矿工并不事先知道金字塔的钻石分布，但可以估计面前两个方块内的钻石数。此时如何行进才可以获得最大数目的金币。

2 数据生成

这里我们使用 numpy 生成进行测试的数据。代码如下：

```
In [1]: %matplotlib inline
        from scipy.interpolate.rbf import Rbf
        import matplotlib.pyplot as plt
        import numpy as np

        x = [1, 9, 4, 5, 1, 4, 1, 9, 1]
        y = [0, 1, 2, 4, 6, 0, 8, 5, 2]
        z = [1]*len(x)

        rbf_adj = Rbf(x, y, z, function='gaussian')

        x_fine = np.linspace(0, 10, 100)
        y_fine = np.linspace(0, 10, 100)

        x_grid, y_grid = np.meshgrid(x_fine, y_fine)

        z_grid = rbf_adj(x_grid.ravel(), y_grid.ravel()).reshape(x_grid.shape) * 10000
        z_grid = z_grid.astype(int)
```

```
for i in range(len(z_grid)):
    for j in range(len(z_grid)):
        if not i + j < len(z_grid):
            z_grid[i][j] = 0

plt.pcolor(x_fine, y_fine, z_grid);
plt.plot(x, y, 'ok');
plt.xlabel('x'); plt.ylabel('y'); plt.colorbar();
```



3 基础问题

3.1 问题性质

考虑我们现在的位置为 (x,y)，很明显我们只能够从 $(x-1,y)$ 或 $(x-1,y-1)$ 两个点来。显然从 $(0,0)$ 到那两个点的路线也必须是能够获得大大价值的路路线，否则我们就可以通过替换为该路线来获得一条比原来总价值更大的路线。而求 $(x-1,y)$ 与 $(x-1,y-1)$ 的最大收益的问题，则是比之前问题规模小的子问题，所以“钻石金字塔”满足最优子结构. 在算法中我们将会重复到达金金金字塔中的结点，即问题满足重复子问题性质。

3.2 递归

考虑我们当前的位置为 (x,y)，我们的下一步无非就是 $(x,y+1)$ 和 $(x+1,y)$ 分别求解即可，代码以及结果如下：

```
In [2]: def get_result_rec(value):
            mask = np.zeros_like(value)
            def _get(x, y):
                if x + y == len(value) - 1:
                    return value[x][y]
                l = _get(x + 1, y)
                r = _get(x, y + 1)
                if l < r:
                    return value[x][y] + r
                else:
                    return value[x][y] + l

            return _get(0,0)

        get_result_rec([[1,2,3],[4,5,0],[6,0,0]])
```

```
Out[2]: 11
```

显然是符合结果的。

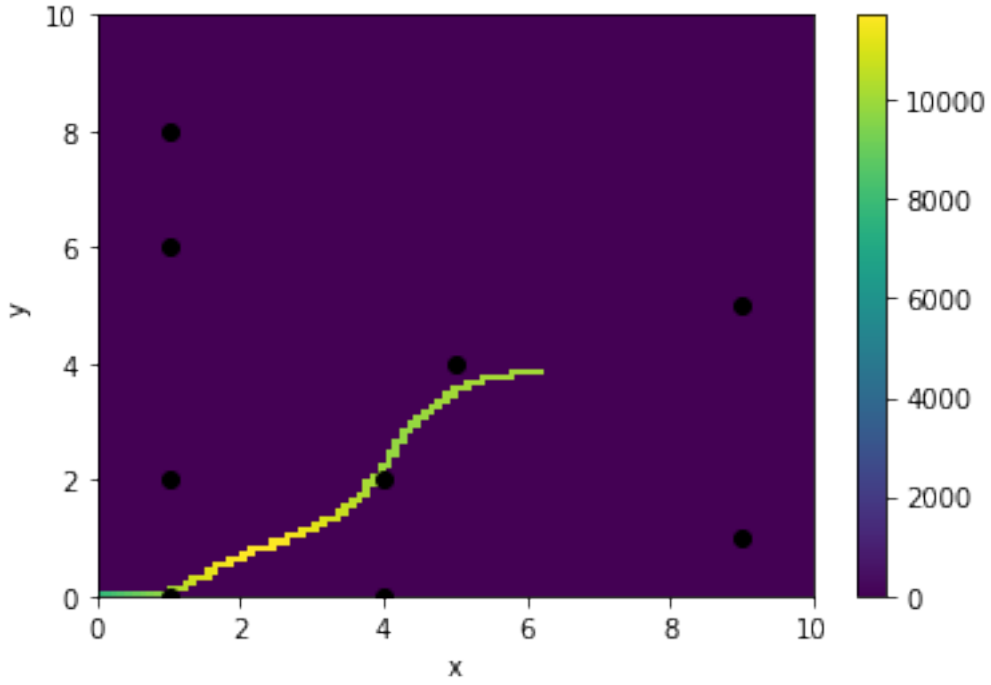
3.3 DP

我们自底向上，逐层遍历，可以得到如下代码：

```
In [3]: def get_result_dp(value):
            dp = []
            route = []
            route_result = np.zeros_like(value)
            n = len(value)
            for i in range(n):
                dp.append([0 for i in range(len(value))])
                route.append([0 for i in range(len(value))])
            for i in range(n):
                dp[i][n - 1 - i] = value[i][n - 1 - i]
            for i in range(n - 1, -1, -1):
                for j in range(i):
                    x = j
                    y = i - 1 - j
                    if dp[x+1][y] > dp[x][y+1]:
                        dp[x][y] = dp[x + 1][y]
                        route[x][y] = 0
                    else:
                        dp[x][y] = dp[x][y + 1]
                        route[x][y] = 1
                    dp[x][y] += value[x][y]

            i = 0
            j = 0
            while i + j < n:
```

```
            route_result[i][j] = value[i][j]
            if route[i][j] == 0:
                i, j = i + 1, j
            else:
                i, j = i, j + 1
        return dp[0][0], route_result

result, route = get_result_dp(z_grid)
plt.pcolor(x_fine, y_fine, route);
plt.plot(x, y, 'ok');
plt.xlabel('x'); plt.ylabel('y'); plt.colorbar();
result
```

Out[3]: 1039114



显然其结果也是最优的。

4 蒙图版

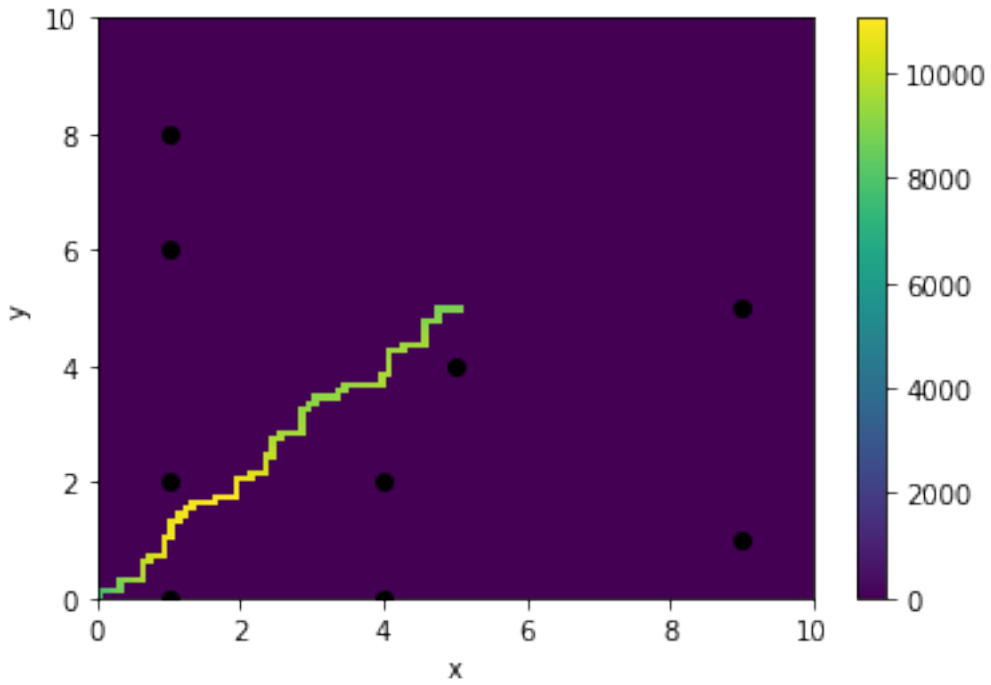
当只能知道下一个解的时候，实际上我们很难做出有效的抉择，这就和人生一样，当你只知道下一步的利弊时你很难做出正确的选择。因此我们在这里只有两种方法：1. 随机选择 2. 只选择下一步最优的

两种算法的代码以及结果如下：

In [4]: # 随机选择

```
def get_result_random(value):
    route = np.zeros_like(value)
    i = 0
    j = 0
    n = len(value)
    result = 0
    while i + j < n:
        route[i][j] = value[i][j]
        result += route[i][j]
        if np.random.randint(1 << 31) % 2 == 0:
            i, j = i + 1, j
        else:
            i, j = i, j + 1
    return result, route

result, route = get_result_random(z_grid)
plt.pcolor(x_fine, y_fine, route);
plt.plot(x, y, 'ok');
plt.xlabel('x'); plt.ylabel('y'); plt.colorbar();
result
```

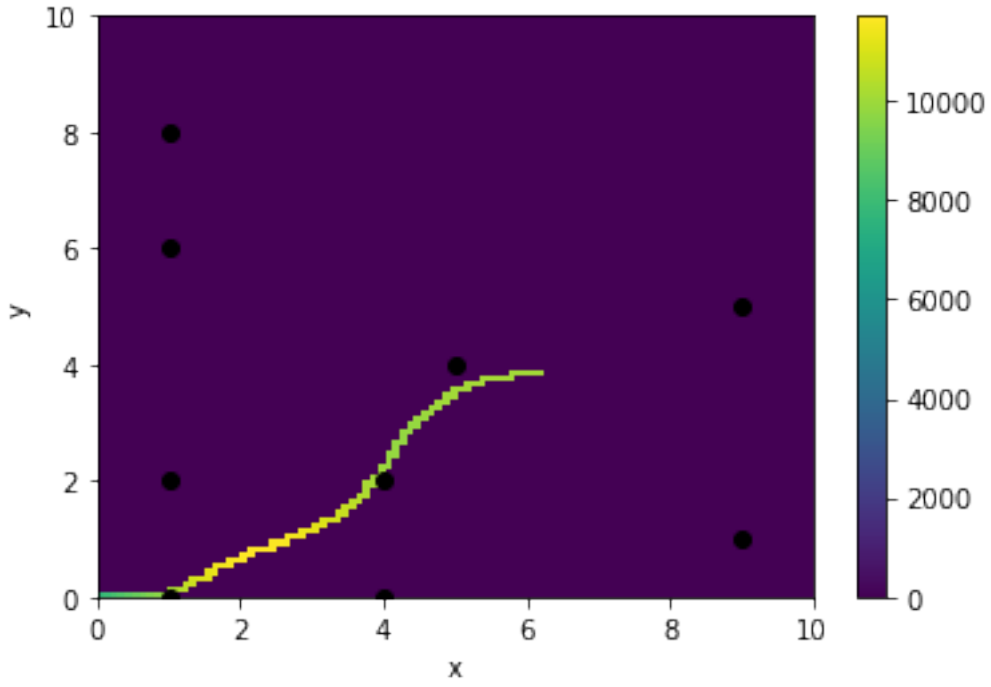
Out[4]: 977712



In [5]: # 下一步最优

```
def get_result_nb(value):
    route = np.zeros_like(value)
    i = 0
    j = 0
    n = len(value)
    result = 0
    while i + j < n:
        route[i][j] = value[i][j]
        result += route[i][j]
        if value[i + 1][j] > value[i][j + 1]:
            i, j = i + 1, j
        else:
            i, j = i, j + 1
    return result, route

result, route = get_result_nb(z_grid)
plt.pcolor(x_fine, y_fine, route);
plt.plot(x, y, 'ok');
plt.xlabel('x'); plt.ylabel('y'); plt.colorbar();
result
```

Out[5]: 1039114



其实结果似乎也不差（