

《算法设计与分析》

课程实验报告



专业： 计算机科学与技术

班级： 2021211304

姓名： 杨晨

学号： 2021212171

1 概述

1.1 实验内容

- 二选一，编程实现下述算法
 - 线性时间选择
 - 平面最近点对
- 利用
 - xx 省会城市 TD-LTE 网络的小区/基站数据，针对线性时间选择、平面最近点对，验证算法正确性，观察分析算法的时间、空间复杂性变化

1.2 开发环境

- Windows10
- PyCharm 2023.2.4 (Professional Edition)

2 实验过程

2.1 线性时间选择

2.1.1 介绍

选择问题涉及在一个无序的包含 n 个元素的列表中找到第 k 小的元素。解决这个问题的朴素方法是对列表进行排序，然后返回第 k 个元素。然而，这种方法使用基于比较的排序算法（如快速排序或归并排序）的时间复杂度为 $O(n \log n)$ 。

线性时间选择算法，也称为快速选择算法，提供了一个高效的解决方案，平均时间复杂度为 $O(n)$ 。它基于快速排序中的划分技术。

2.1.2 算法描述

快速选择算法通过从列表中选择一个枢轴元素，并围绕枢轴将元素进行划分，使得所有小于枢轴的元素在其左侧，所有大于枢轴的元素在其右侧。这个过程类似于快速排序中的划分步骤。

中位数法选择枢轴是一种优化策略，它确保了良好的枢轴选择，从而避免了最坏情况的发生。该方法的步骤如下：

1. 将列表划分为大小为 5 的子列表（最后一个子列表的大小可以小于 5）。
2. 对每个子列表进行排序。
3. 从每个子列表中选择中位数，将这些中位数组成一个新的列表。
4. 递归地应用快速选择算法来选择新列表的中位数作为枢轴。

算法继续使用快速选择的步骤，根据枢轴将列表划分为左、右两个部分，并根据 k 的值进行递归搜索。如果枢轴元素是第 k 个元素，我们找到了所需的值并返回它。否则，我们根据 k 是小于还是大于枢轴的索引，在左侧或右侧划分上递归应用算法。

快速选择算法的伪代码如算法1所示。

Algorithm 1 快速选择算法

function QUICKSELECT($A, low, high, k$)

$pivot \leftarrow$ 中位数法选择枢轴 (A)

 ▷ 使用中位数法选择一个枢轴元素

 交换 $A[low]$ 和 $pivot$

 ▷ 将枢轴元素放到开头

$mid \leftarrow$ PARTITION($A, low, high$)

 ▷ 调用 Partition 函数

if $mid - low + 1 == k$ **then**

return $A[mid]$

 ▷ 找到第 k 个元素

else if $mid - low + 1 > k$ **then**

return QUICKSELECT($A, low, mid - 1, k$)

 ▷ 在左侧划分中递归搜索

else

return QUICKSELECT($A, mid + 1, high, k - (mid - low + 1)$)

 ▷ 在右侧划分中递归搜索

function PARTITION($A, low, high$)

$pivot \leftarrow A[low]$

 ▷ 以列表开头元素划分

$i \leftarrow low + 1$

$j \leftarrow high$

while True do

 ▷ 按照枢轴元素进行划分，同快速排序

while $i \leq j$ **并且** $A[i] \leq pivot$ **do**

$i \leftarrow i + 1$

while $i \leq j$ **并且** $A[j] \geq pivot$ **do**

$j \leftarrow j - 1$

if $i > j$ **then**

break

 交换 $A[i]$ 和 $A[j]$

 交换 $A[low]$ 和 $A[j]$

return j

 ▷ 返回枢轴元素下标

2.1.3 分析

快速选择算法的平均时间复杂度为 $O(n)$ ，通过使用中位数法选择枢轴，可以避免最坏情况的发生，确保算法的时间复杂度为 $O(n)$ 。中位数法选择枢轴的时间复杂度为 $O(n)$ ，因为它涉及对子列表进行排序和选择中位数。

选择问题的最坏情况发生在每次划分都产生极度不平衡的子列表时，这时快速选择算法的时间复杂度可能达到 $O(n^2)$ 。然而，通过使用中位数法选择枢轴，我们可以减少最坏情况的概率，并获得平均时间复杂度为 $O(n)$ 的性能。

2.1.4 运行结果

最小的元素为： 103.075

递归最大层次为： 7

第5小的元素为： 126.096

递归最大层次为： 7

第50小的元素为： 208.475

递归最大层次为： 7

最大的元素为： 2735.798

递归最大层次为： 7

通过排序验证结果的正确性：

最小的元素为： 103.075

第5小的元素为： 126.096

第50小的元素为： 208.475

最大的元素为： 2735.798

2.1.5 结论

线性时间选择算法，即快速选择算法，为选择问题提供了高效的解决方案。通过利用快速排序中的划分技术和中位数法选择枢轴，它实现了平均时间复杂度为 $O(n)$ ，因此在查找无序列表中第 k 小的元素时是一个很好的选择。中位数法选择枢轴可以避免最坏情况的发生，确保算法的性能始终为 $O(n)$ 。快速选择算法在实践中被广泛应用于各种需要选择问题的场景中，提供了高效的解决方案。

2.2 最近平面点对

2.2.1 介绍

给定平面上的 n 个点 $P = \{p_0, p_2, \dots, p_{n-1}\}$ ，其中每个点 p_i 的坐标表示为 (x_i, y_i) 。最近平面点对问题要求找到两个点 p_i 和 p_j ，使得它们的欧氏距离 $d(p_i, p_j)$ 最小，即 $d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ 。

2.2.2 算法描述

最近平面点对算法采用分治法的思想来提高效率。分治法的基本思路是，将大问题转化为类似的小问题，解决小问题后通过合并解决大问题。在本题中，我们要解决的问题就是计算标号为 $1, 2 \dots (n-1)$ 的这 n 个点的两点间最近距离。我们设计算标号为 $l, l+1 \dots (r-1)$ 的点的两点间最近距离为 $f(l, r)$ ，最终目标即求 $f(0, n)$ 。要想解决 $f(l, r)$ ，可以考虑先解决 $f(l, \lfloor \frac{l+r}{2} \rfloor)$ 和 $f(\lfloor \frac{l+r}{2} \rfloor, r)$

容易发现，当 $r = l + 1$ 时不需要做任何处理（此时返回 ∞ ）。关键在于，如何合并两个子问题的结果。

1. 记 d 为左右区间的最小距离，找到左右两部分的中线 $mid = l + \frac{r-l}{2}$ ，此时只需要在横坐标 $x \in (mid - d, mid + d)$ 的区间中找即可
2. 对所有横坐标 $x \in (mid - d, mid + d)$ 的点按照 y 排序，我们只需要考虑 $y_j - y_i < d (i < j)$ 的点对

所以，合并时，中线附近的点，只需要考虑的范围是一个长为 $2d$ 宽为 d 的矩形。容易证明，这个区域内最多扫描六个点



$$\sqrt{\frac{1^2}{2} + \frac{2^2}{3}} = \frac{5}{6} < 1$$

最近平面点对算法的伪代码如算法2所示

2.2.3 分析

最近平面点对算法的时间复杂度为 $O(n \log n)$ ，空间复杂度为 $O(n)$ 。以下是对算法性能的进一步分析：

- 算法的主要开销在于对点集的排序以及递归过程中的计算。
- 排序点集的时间复杂度为 $O(n \log n)$ ，可以使用快速排序或归并排序等常见排序算法实现。
- 递归过程中，每次将点集划分为两部分，并在每个子问题中计算最近点对，因此总共需要进行 $O(\log n)$ 层递归。
- 在每层递归中，计算最近点对的时间复杂度为 $O(n)$ 。

Algorithm 2 NearestPair

function MERGE(*stations*, *l*, *mid*, *r*) ▷ 合并两个有序数组, 区间 $[l, mid)$ 和 $[mid, r)$, 按照 y 坐标排序

merged \leftarrow sorted(*stations*[*l*:*r*], λ station : station.y)

stations[*l*:*r*] \leftarrow merged

function NEARESTPAIR(*stations*, *l*, *r*, *min_dis*, *lesser_dis*) ▷ 求解最近点对, 区间为 $[l, r)$

if $r - l \leq 1$ **then**

return *min_dis*, *lesser_dis*

mid $\leftarrow l + (r - l) \div 2$ ▷ 递归求解左右两边的最近点对

min_dis, *lesser_dis* \leftarrow NearestPair(*stations*, *l*, *mid*, *min_dis*, *lesser_dis*)

min_dis, *lesser_dis* \leftarrow NearestPair(*stations*, *mid*, *r*, *min_dis*, *lesser_dis*)

Merge(*stations*, *l*, *mid*, *r*) ▷ 合并两个有序数组, 按照 y 坐标排序

mid_line \leftarrow *stations*[*mid*].x ▷ 中线

selected \leftarrow []

for *i* $\leftarrow l$ to *r* **do**

if $|\text{stations}[i].x - \text{mid_line}| < \text{min_dis}[0]$ **then**

selected.append(*stations*[*i*])

for *i* $\leftarrow 0$ to len(*selected*) - 1 **do**

for *j* $\leftarrow i + 1$ to len(*selected*) **do**

if $\text{selected}[j].y - \text{selected}[i].y \geq \text{min_dis}[0]$ **then**

 break

dis \leftarrow calculate_distance(*selected*[*i*].y, *selected*[*i*].x, *selected*[*j*].y, *selected*[*j*].x)

if *dis* = 0 **then**

 continue

if *dis* < *min_dis*[0] **then** ▷ 更新最近点对

lesser_dis \leftarrow *min_dis*

min_dis \leftarrow (*dis*, *selected*[*i*].eNodeB_id, *selected*[*j*].eNodeB_id)

else if *min_dis*[0] < *dis* < *lesser_dis*[0] **then** ▷ 更新次近点对

lesser_dis \leftarrow (*dis*, *selected*[*i*].eNodeB_id, *selected*[*j*].eNodeB_id)

return *min_dis*, *lesser_dis*

2.2.4 运行结果

最近点对距离为： 1.278655 , eNodeB_id分别为： 567389 566803

次近点对距离为： 1.673381 , eNodeB_id分别为： 567222 566784

通过遍历求解的结果：

最近点对距离为： 1.278655 , eNodeB_id分别为： 566803 567389

次近点对距离为： 1.673381 , eNodeB_id分别为： 566784 567222

2.2.5 结论

最近平面点对算法是一种高效的方法，用于解决平面上最接近的点对问题。通过使用分治法和合适的数据结构，可以在较短的时间内找到最近点对。该算法在计算几何学和计算机图形学等领域有广泛应用。

3 附录：完整代码

3.1 线性时间选择

```
import pandas as pd

def bubble_sort(arr, low, high):
    """
    冒泡排序
    :param arr: 待排序列表
    :param low: 左边界
    :param high: 右边界
    :return: 从低到高排序的列表
    """
    n = high - low + 1
    for i in range(n):
        is_swapped = False
        for j in range(n - i - 1):
            if arr[j + low] > arr[j + 1 + low]:
                arr[j + low], arr[j + 1 + low] = arr[j + 1 + low], arr[j + low]
                is_swapped = True
        if not is_swapped:
            break

# 全局变量，记录选择划分过程的递归层次
recursion_level = 0
```

```

def clear_level():
    """
    将递归层次清零
    :return:
    """
    global recursion_level
    recursion_level = 0

def linear_select(arr, low, high, k, current_level=1):
    """
    线性时间选择算法
    :param arr: 待划分数组
    :param low: 左边界
    :param high: 右边界
    :param k: 选择的第k小的元素
    :return:
    """
    global recursion_level
    recursion_level = max(recursion_level, current_level)
    if high - low + 1 < 20:
        # 如果数组长度小于20, 则直接排序
        bubble_sort(arr, low, high)
        return arr[low + k - 1]

    # 5个元素一组, 分别排序
    for i in range(low, high - 4, 5):
        bubble_sort(arr, i, i + 4)
        # 将中位数放到数组最前面
        arr[low + (i - low) // 5], arr[i + 2] = arr[i + 2], arr[low + (i - low) // 5]

    # 得到中位数的中位数
    cnt = (high - low + 1) // 5
    pivot = linear_select(arr, low, low + cnt - 1, cnt // 2 + 1, current_level + 1)
    # 得到pivot的下标
    pivot = arr.index(pivot)
    # 将pivot放到数组最前面
    arr[low], arr[pivot] = arr[pivot], arr[low]
    # 一分为三
    pivot = partition_three(arr, low, high)
    # 递归寻找
    if pivot - low + 1 == k:
        return arr[pivot]
    elif pivot - low + 1 > k:
        return linear_select(arr, low, pivot - 1, k, current_level + 1)
    else:
        return linear_select(

```



```

        arr, pivot + 1, high, k - (pivot - low + 1), current_level + 1
    )

def partition_three(arr, low, high):
    """
    一分为三,基准元素为数组第一个元素
    :param arr:待划分数组
    :param low: 左边界
    :param high: 右边界
    :return: 划分后基准元素的下标
    """
    pivot = arr[low]
    # 将数组分为三部分
    i = low + 1
    j = high
    while True:
        while i <= j and arr[i] <= pivot:
            i += 1
        while i <= j and arr[j] >= pivot:
            j -= 1
        if i > j:
            break
        arr[i], arr[j] = arr[j], arr[i]
    arr[low], arr[j] = arr[j], arr[low]
    return j

if __name__ == "__main__":
    # 读取基站数据文件
    data = pd.read_excel(
        r"C:\Users\Administrator\Desktop\算法设计与分析-编程作业-第2章□分治
        -2023\02-1□1033个基站数据.xls"
    )
    # 删除未命名的列
    data = data.loc[:, ~data.columns.str.contains("^Unnamed")]

    # 提取k-dist列的数据
    k_dist_values = data["K_DIST"].tolist()
    k_dist_copy = k_dist_values.copy() # 复制一份用于验证结果的正确性

    # 选择最小的元素
    min_dist_values = linear_select(k_dist_values, 0, len(k_dist_values) - 1, 1)
    print("最小的元素为: ", min_dist_values)
    print("递归最大层次为: ", recursion_level, "\n")

    # 选择第5小的元素

```

```

clear_level()
min_dist_values = linear_select(k_dist_values, 0, len(k_dist_values) - 1, 5)
print("第5小的元素为: ", min_dist_values)
print("递归最大层次为: ", recursion_level, "\n")

# 选择第50小的元素
clear_level()
min_dist_values = linear_select(k_dist_values, 0, len(k_dist_values) - 1, 50)
print("第50小的元素为: ", min_dist_values)
print("递归最大层次为: ", recursion_level, "\n")

# 选择最大的元素
clear_level()
min_dist_values = linear_select(
    k_dist_values, 0, len(k_dist_values) - 1, len(k_dist_values)
)
print("最大的元素为: ", min_dist_values)
print("递归最大层次为: ", recursion_level, "\n")

# 验证结果的正确性
k_dist_copy.sort()
print("通过排序验证结果的正确性: ")
print("最小的元素为: ", k_dist_copy[0])
print("第5小的元素为: ", k_dist_copy[4])
print("第50小的元素为: ", k_dist_copy[49])
print("最大的元素为: ", k_dist_copy[-1])

```

3.2 最近平面点对

```

import numpy as np
import pandas as pd

def degree_to_radian(degree):
    """
    将给定的经纬度转化为弧度
    :param degree: 1个输入(经纬度)
    :return: 转化为的弧度
    """
    return degree * np.pi / 180

def calculate_distance(lat1, lon1, lat2, lon2):
    """
    距离公式
    :param lat1: 纬度1

```

```

:param lon1: 经度1
:param lat2: 纬度2
:param lon2: 经度2
:return: 距离/m,保留6位小数
"""
if abs(lat1 - lat2) < 1e-6 and abs(lon1 - lon2) < 1e-6:
    return 0
rad_lat1 = degree_to_radian(lat1)
rad_lon1 = degree_to_radian(lon1)
rad_lat2 = degree_to_radian(lat2)
rad_lon2 = degree_to_radian(lon2)
# R为赤道半径/m
R = 6378.137 * 1000
dis = R * np.arccos(
    np.cos(rad_lat1) * np.cos(rad_lat2) * np.cos(rad_lon1 - rad_lon2)
    + np.sin(rad_lat1) * np.sin(rad_lat2)
)
return round(dis, 6)

class Station:
    def __init__(self, eNodeB_id, latitude, longitude):
        self.eNodeB_id = eNodeB_id
        self.x = longitude
        self.y = latitude

def merge(stations, l, mid, r):
    """
    合并两个有序数组，区间[l,mid)和[mid,r)，但是按照y坐标排序
    :param stations: 基站列表
    :param l: 左边界
    :param mid: 中间位置
    :param r: 右边界
    """
    merged = sorted(stations[l:r], key=lambda station: station.y)
    stations[l:r] = merged

def nearest_pair(stations, l, r, min_dis, lesser_dis):
    """
    求解最近点对，区间为[l,r)
    :param stations: 基站列表
    :param l: 左边界
    :param r: 右边界
    :param min_dis: 最近距离（距离，eNodeB_id1, eNodeB_id2）
    :param lesser_dis: 次近距离（距离，eNodeB_id1, eNodeB_id2）

```

```

: return: 最近距离 (距离, eNodeB_id1, eNodeB_id2), 次近距离 (距离, eNodeB_id1,
        eNodeB_id2)
"""
if r - l <= 1:
    return min_dis, lesser_dis
mid = 1 + (r - l) // 2
# 递归求解左右两边的最近点对
min_dis, lesser_dis = nearest_pair(stations, l, mid, min_dis, lesser_dis)
min_dis, lesser_dis = nearest_pair(stations, mid, r, min_dis, lesser_dis)
# 合并两个有序数组, 但是按照y坐标排序
merge(stations, l, mid, r)
# 选取中间区域的点
mid_line = stations[mid].x # 中线
selected = []
for i in range(l, r): # 选取中线左右两边距离中线小于min_dis的点
    if abs(stations[i].x - mid_line) < min_dis[0]:
        selected.append(stations[i])
# 计算最近点对
for i in range(len(selected)):
    for j in range(i + 1, len(selected)):
        if selected[j].y - selected[i].y >= min_dis[0]: # 剪枝, y距离大于
            min_dis的点不用计算
            break
        dis = calculate_distance(
            selected[i].y, selected[i].x, selected[j].y, selected[j].x
        )
        if dis == 0: # 剪枝, 距离为0的点对不用计算
            continue
        if dis < min_dis[0]: # 更新最近点对
            lesser_dis = min_dis
            min_dis = (dis, selected[i].eNodeB_id, selected[j].eNodeB_id)
        elif min_dis[0] < dis < lesser_dis[0]: # 更新次近点对
            lesser_dis = (dis, selected[i].eNodeB_id, selected[j].eNodeB_id)

    return min_dis, lesser_dis

if __name__ == "__main__":
    # 读取基站数据文件
    data = pd.read_excel(
        r"C:\Users\Administrator\Desktop\算法设计与分析-编程作业-第2章_分治
        -2023\02-1_1033个基站数据.xls"
    )
    # 删除未命名的列
    data = data.loc[:, ~data.columns.str.contains("^Unnamed")]

    eNodeB_id = data["ENODEBID"].tolist()

```

```

longitude = data["LONGITUDE"].tolist()
latitude = data["LATITUDE"].tolist()

# 创建对象的列表
stations = [
    Station(eNodeB_id, lon, lat)
    for eNodeB_id, lon, lat in zip(eNodeB_id, longitude, latitude)
]
stations_copy = stations.copy() # 复制一份用于验证结果的正确性

min_dis = (1e10, -1, -1)
lesser_dis = (1e10, -1, -1)

stations.sort(key=lambda station: station.x)

# 求解最近点对
min_dis, lesser_dis = nearest_pair(stations, 0, len(stations), min_dis,
    lesser_dis)
print("最近点对距离为: ", min_dis[0], ", eNodeB_id分别为: ", min_dis[1],
    min_dis[2])
print("次近点对距离为: ", lesser_dis[0], ", eNodeB_id分别为: ", lesser_dis[1],
    lesser_dis[2])

# 验证结果的正确性
min_dis_ = 1e10
lesser_dis_ = 1e10
min_id1, min_id2, lesser_id1, lesser_id2 = -1, -1, -1, -1
for i in range(len(stations_copy)):
    for j in range(i + 1, len(stations_copy)):
        dis = calculate_distance(
            stations_copy[i].y,
            stations_copy[i].x,
            stations_copy[j].y,
            stations_copy[j].x,
        )
        if dis == 0: # 剪枝, 距离为0的点对不用计算
            continue
        if dis < min_dis_:
            lesser_dis_ = min_dis_
            lesser_id1, lesser_id2 = min_id1, min_id2
            min_dis_ = dis
            min_id1, min_id2 = (
                stations_copy[i].eNodeB_id,
                stations_copy[j].eNodeB_id,
            )
        elif min_dis_ < dis < lesser_dis_:
            lesser_dis_ = dis

```

```
        lesser_id1, lesser_id2 = (  
            stations_copy[i].eNodeB_id,  
            stations_copy[j].eNodeB_id,  
        )  
print("\n通过遍历求解的结果: ")  
print("最近点对距离为: ", min_dis_, ", eNodeB_id分别为: ", min_id1, min_id2)  
print("次近点对距离为: ", lesser_dis_, ", eNodeB_id分别为: ", lesser_id1,  
      lesser_id2)
```