19220422时子延-数据结构课设报告

一、必做题

> Quote
>
> 编程实现希尔、快速、堆排序、归并排序算法。要求随机产生10000个数据存入磁盘文件，然后读入数据文件，分别采用不同的排序方法进行排序，并将结果存入文件中。

算法思想描述

- 随机生成数据
- 文件读写
- 希尔排序
- 快速排序
- 堆排序
- 归并排序

程序结构

本项目使用`Makefile`管理，使用`g++`作为编译器，采用`c++11`标准。

运行`make`,`./a.out` 即可得到结果。

```
(base) szy@SzyAir solution1 % tree
.
├── Makefile
├── a.out
├── data.txt
├── main.cpp
├── main.o
├── res_heap.txt
├── res_merge.txt
├── res_quick.txt
└── res_shell.txt
```

`make clean`后回到原始文件

```
(base) szy@SzyAir solution1 % tree
.
├── Makefile
└── main.cpp
```

```
0 directories, 2 files
```

- `Makefile`

```
# Makefile

# Compiler
CC = g++

# Compiler flags
CFLAGS = -std=c++11 -Wall

# Source files
SRCS = main.cpp

# Object files
OBJS = $(SRCS:.cpp=.o)

# Executable
TARGET = a.out

all: $(TARGET)

$(TARGET): $(OBJS)
    $(CC) $(CFLAGS) $(OBJS) -o $(TARGET)

%.o: %.cpp
    $(CC) $(CFLAGS) -c $< -o $@

clean:
    rm -f $(OBJS) $(TARGET) *.txt
```

- `main.cpp`

```
#include<iostream>
#include<fstream>

using namespace std;

int a[10001];
int N=10000;

//随机产生10000个数据存入磁盘文件
void RandomData(const string& filePath)
{
    ofstream fout(filePath);
    for(int i = 0; i < 10000; i++)
    {
        fout << rand() << endl;
    }
    fout.close();
}

//读取data1.txt文件
void ReadData(const string& filePath)
{
    ifstream fin(filePath);
    int n;
    int i=0;
    while(fin >> n)
    {
        a[i++] = n;
    }
    //cout<<"i = "<<i<<endl;
    fin.close();
}

//随机产生10000个数据存入磁盘文件
void test_readData()
{
    ofstream fout("data2.txt");
    for(int i = 0; i < 10000; i++)
    {
        fout << a[i] << endl;
    }
    fout.close();
}

//希尔排序
void shellsort(int a[],int n){
    for(int gap = n/2; gap > 0; gap /= 2){
        for(int i = gap; i < n; i++){
            int temp = a[i];
            int j;
            for(j = i-gap; j >= 0 && a[j] > temp; j -= gap){
                a[j+gap] = a[j];
            }
            a[j+gap] = temp;
        }
    }
    ofstream fout("res_shell.txt");
    for(int i = 0; i < 10000; i++)
    {
        fout << a[i] << endl;
    }
    fout.close();
}


void quicksort(int a[],int n){
    if(n <= 1) return;
    int pivot = a[0];
    int i = 1;
    int j = n-1;
    while(i <= j){
        while(i <= j && a[i] <= pivot) i++;
        while(i <= j && a[j] > pivot) j--;
        if(i < j) swap(a[i],a[j]);
    }
    swap(a[0],a[j]);
    quicksort(a,j);
    quicksort(a+j+1,n-j-1);
}

void QuickSort(int a[],int n){
    quicksort(a,n);
    ofstream fout("res_quick.txt");
    for(int i = 0; i < 10000; i++)
    {
        fout << a[i] << endl;
    }
    fout.close();
}

void swap(int *a,int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

void max_heapify(int arr[], int start, int end) {
    //建立父节点指标和子节点指标
    int dad = start;
    int son = dad * 2 + 1;
    while (son <= end) { //若子节点指标在范围内才做比较
        if (son + 1 <= end && arr[son] < arr[son + 1]) //先比较两个子节点大小，选择
            son++;
        if (arr[dad] > arr[son]) //如果父节点大于子节点代表调整完毕，直接跳出函数
            return;
        else { //否则交换父子内容再继续子节点和孙节点比较
            swap(&arr[dad], &arr[son]);
            dad = son;
            son = dad * 2 + 1;
        }
    }
}

// 堆排序，最小堆
void heapsort(int arr[],int len){
    int i;
    //初始化，i从最后一个父节点开始调整
    for (i = len / 2 - 1; i >= 0; i--)
        max_heapify(arr, i, len - 1);
    //先将第一个元素和已排好元素前一位做交换，再从新调整，直到排序完毕
    for (i = len - 1; i > 0; i--) {
        swap(&arr[0], &arr[i]);
        max_heapify(arr, 0, i - 1);
    }
    ofstream fout("res_heap.txt");
    for(int i = 0; i < 10000; i++)
    {
        fout << arr[i] << endl;
    }
    fout.close();
}

//归并排序

void mergesort(int a[],int n,int l,int r){
    if(l == r) return;
    int mid = (l+r)/2;
    mergesort(a,n,l,mid);
    mergesort(a,n,mid+1,r);
    int i = l,j = mid+1,k = 0;
    int temp[r-l+1];
    while(i <= mid && j <= r){
        if(a[i] <= a[j]) temp[k++] = a[i++];
        else temp[k++] = a[j++];
    }
    while(i <= mid) temp[k++] = a[i++];
    while(j <= r) temp[k++] = a[j++];
    for(int i = l;i <= r;i++) a[i] = temp[i-l];
}

void MergeSort(int a[],int n){
    mergesort(a,n,0,n-1);
    ofstream fout("res_merge.txt");
    for(int i = 0; i < 10000; i++)
    {
        fout << a[i] << endl;
    }
    fout.close();
}

int main()
{
    RandomData("data.txt");

    ReadData("data.txt");
    //test_readData();
    shellsort(a,N);

    ReadData("data.txt");
    QuickSort(a,N);

    ReadData("data.txt");
    heapsort(a,N);

    ReadData("data.txt");
    MergeSort(a,N);



    return 0;
}
```

测试结果

完成任务，详见`res_*.txt`

收获与体会

复习了随机数，文件I/O，排序算法。实践了`Makefile`，完成了课程设计任务。

二、选做题

> Quote
>
> **5. 求解最短路径**
>
> 设有N（$N>10$）个城市之间的交通图，假设任意两个城市之间不一定有直接交通线路，权表示乘车时间。要求事先将交通图信息将存入磁盘文件中，求从某城市出发到其他城市的最少乘车时间和乘车路线。要求将结果以图形方式在屏幕上输出。

算法思想描述

“从某城市出发到其他城市的最少乘车时间和乘车路线”可以化归为求图中单源最短路径问题。使用Dijstra算法。

“事先将交通图信息将存入磁盘文件中”，涉及图的元数据存储。考虑到`G=<E,V>`，采用读入`NumberOfEdge`,`NumberOfVertex`,然后通过读入`v_1,v_2,w_12`的方法逐行读入每条边的信息，构建邻接表表示图。

“要求将结果以图形方式在屏幕上输出。” 仅仅使用C++和命令行恐怕有点困难。参考一下内容，该用Python完成。

- Python 可交互的网络图可视化工具 - 知乎 (zhihu.com)
- Graph | NetworkX 入门教程 - 知乎 (zhihu.com)
- NetworkX — NetworkX documentation

程序结构

节点

```
class Vertex:
    def __init__(self, name: str):
        self.name = name
        self.next = []
```

边

```
class Edge:
    def __init__(self, start, end, weight):
        self.start = start
        self.end = end
        self.weight = weight
```

图

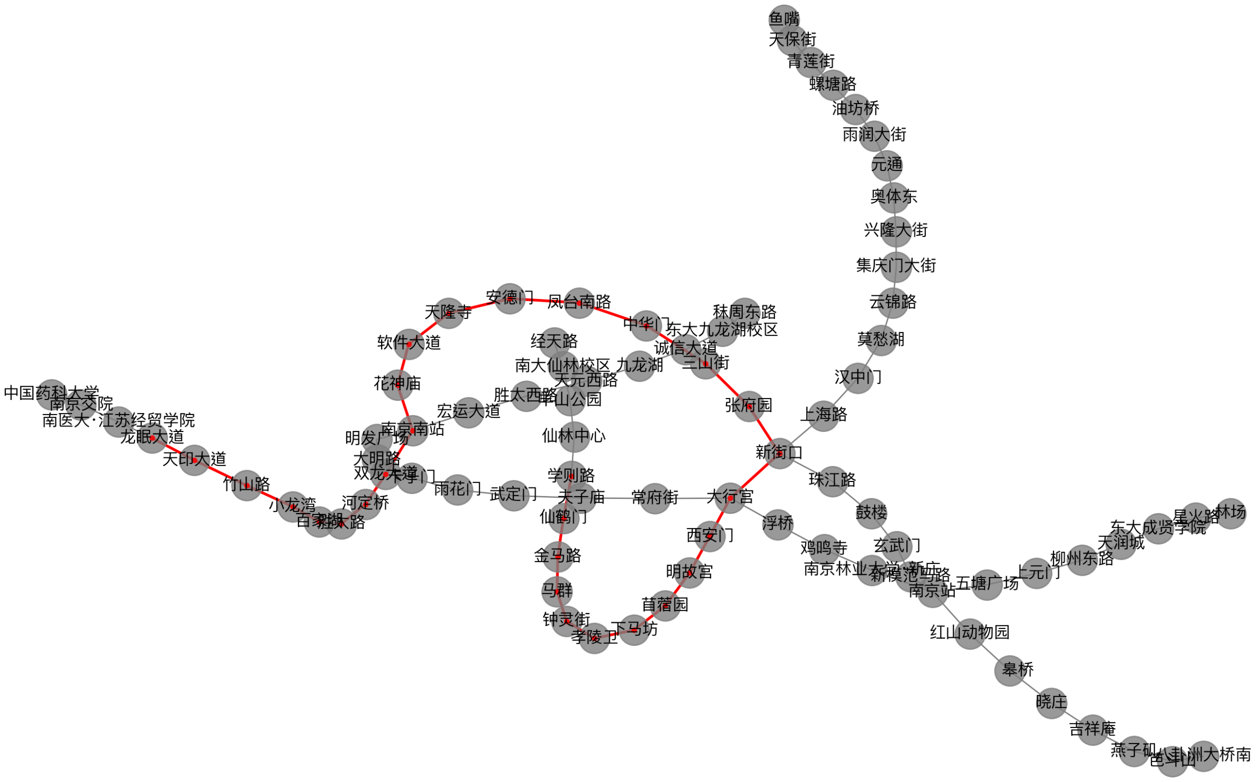
```
class NanjingMetro:
    def __init__(self):
        self.Vertexes = []
        self.Edges = []
        self.G = nx.Graph()

    def ImportGraph(self,filename):
        with open(filename,'r') as f:
            print("Importing "+filename)

            lines = f.readlines()

            AmountOfGraph = int(lines[0])
            print("Amount of Graph:",AmountOfGraph)

            lines = lines[1:]

            while(AmountOfGraph>0):
                AmountOfGraph -= 1
                NumberOfEdge = int(lines[0])
                print("Amount of Edge:",NumberOfEdge)
                lines = lines[1:]
                for i in range(NumberOfEdge):
                    start, end, weight = lines[i].split(',')
                    if self.checkVertex(start) == False:
                        self.Vertexes.append(Vertex(start))
                        # self.G.add_node(start,{"color":"gree"})
                    if self.checkVertex(end) == False:
                        self.Vertexes.append(Vertex(end))


                    self.setVertex(start).next.append(end)
                    self.setVertex(end).next.append(start)

                    edge = Edge(start, end, int(weight))
                    if edge not in self.Edges:
                        self.Edges.append(edge)
                    edge = Edge(end, start, int(weight))
                    if edge not in self.Edges:
                        self.Edges.append(edge)
                lines = lines[NumberOfEdge:]
            print("Importing finished")



    def getNumberOfVertexes(self):
        return len(self.Vertexes)
    def getNumberOfEdges(self):
        return len(self.Edges)/2  #无向图

    def checkVertex(self,name):
        for i in self.Vertexes:
            if i.name == name:
                return True
        return False
    def setVertex(self,name):
        for i in self.Vertexes:
            if i.name == name:
                return i
    def getVertexIndex(self,name):
        for i in range(len(self.Vertexes)):
            if name == self.Vertexes[i].name:
                return i
    def setEdge(self,start,end):
        for i in self.Edges:
            if i.start == start and i.end == end:
                return i

    def print_adjacency_list(self):
        # list = [i.name for i in self.Vertexes]
        # print(list)
        for vertex in self.Vertexes:
            print(vertex.name,":",end=" ")
            for i in vertex.next:
                #print(i,self.setEdge(vertex.name,i).weight,end=" ")
                print(i,end=" ")
            print()


    # 求A站到B站到最短路线 龙眠大道——学则路
    # 给出Dijstra算法
    def ShortPath(self,start:str,end:str):
        parent = {} #节点关系，用于回溯
        distance = {} #起点到各个节点的距离
        # init
        for i in self.Vertexes:
            distance[i.name] = float('inf')
        distance[start] = 0

        queue = []
        heapq.heappush(queue,[0,start])
        while len(queue) > 0:
            d, v = heapq.heappop(queue)
            if d > distance[v]:
                continue
            for i in self.setVertex(v).next:
                if distance[i] > distance[v] + self.setEdge(v,i).weight:
                    distance[i] = distance[v] + self.setEdge(v,i).weight
                    parent[i] = v
                    heapq.heappush(queue,[distance[i],i])
        print("Distance from",start,"to",end,"=",distance[end])
        path = [end]
        while path[-1] != start:
            path.append(parent[path[-1]])
        path.reverse()
        print(path)
        self.showPath(path,start,end)






    def showPath(self, path,start,end):
        G = nx.Graph()
        for vertex in self.Vertexes:
            G.add_node(vertex.name)
            for i in vertex.next:
                G.add_edge(vertex.name, i, weight=self.setEdge(vertex.name, i).

        pos = nx.spring_layout(G)
        nx.draw_networkx_nodes(G, pos, node_color='gray', node_size=500, alpha=
        nx.draw_networkx_edges(G, pos, edge_color='gray')

        path_edges = [(path[i], path[i+1]) for i in range(len(path)-1)]
        nx.draw_networkx_edges(G, pos, edgelist=path_edges, edge_color='red', w

        for i in path:
            nx.draw_networkx_nodes(G, pos, nodelist=[i], node_color='red', node_

        font_family = None
        for f in ['PingFang HK','Microsoft Sans Serif','SimHei', 'Microsoft YaH
            if f in plt.rcParams['font.family']:
                font_family = f
                break
        nx.draw_networkx_labels(G, pos, font_family='PingFang HK', font_size=12
        plt.axis('off')
        plt.show()
        plt.savefig(start,"到",end,"最短路径.png")

```

测试结果

测试代码如下：

```
if __name__ == '__main__':
    filename = "./南京地铁.txt"
    # filename = "./南京地铁E.txt"
    nm = NanjingMetro()
    nm.ImportGraph(filename)
    nm.print_adjacency_list()
    nm.ShortPath("龙眠大道","学则路")
```

结果：



收获与体会

复习了上学期学的邻接表表示图的应用与Dijstar算法，实践了通过文件读写来构建一个图。学习了networkx这一将图数据可视化的python package。首次基于真实的图数据（南京地铁交通线路图）来应用学到的知识。对图的应用与效果有了深刻的体会。

程序清单

```
(soinn) szy@SzyAir 数据结构课设 % tree
.
```

```
├── solution1
│   ├── Makefile
│   └── main.cpp
├── solution2
│   ├── main.py
│   ├── old
│   │   ├── Makefile
│   │   ├── main.old.py
│   │   ├── nx.ipynb
│   │   ├── nx.py
│   │   ├── simple_path.png
│   │   ├── solution2.cpp
│   │   ├── 南京地铁 copy.txt
│   │   └── 南京地铁E.txt
│   ├── 南京地铁.jpeg
│   └── 南京地铁.txt
└── 数据结构-课程设计题.docx
```