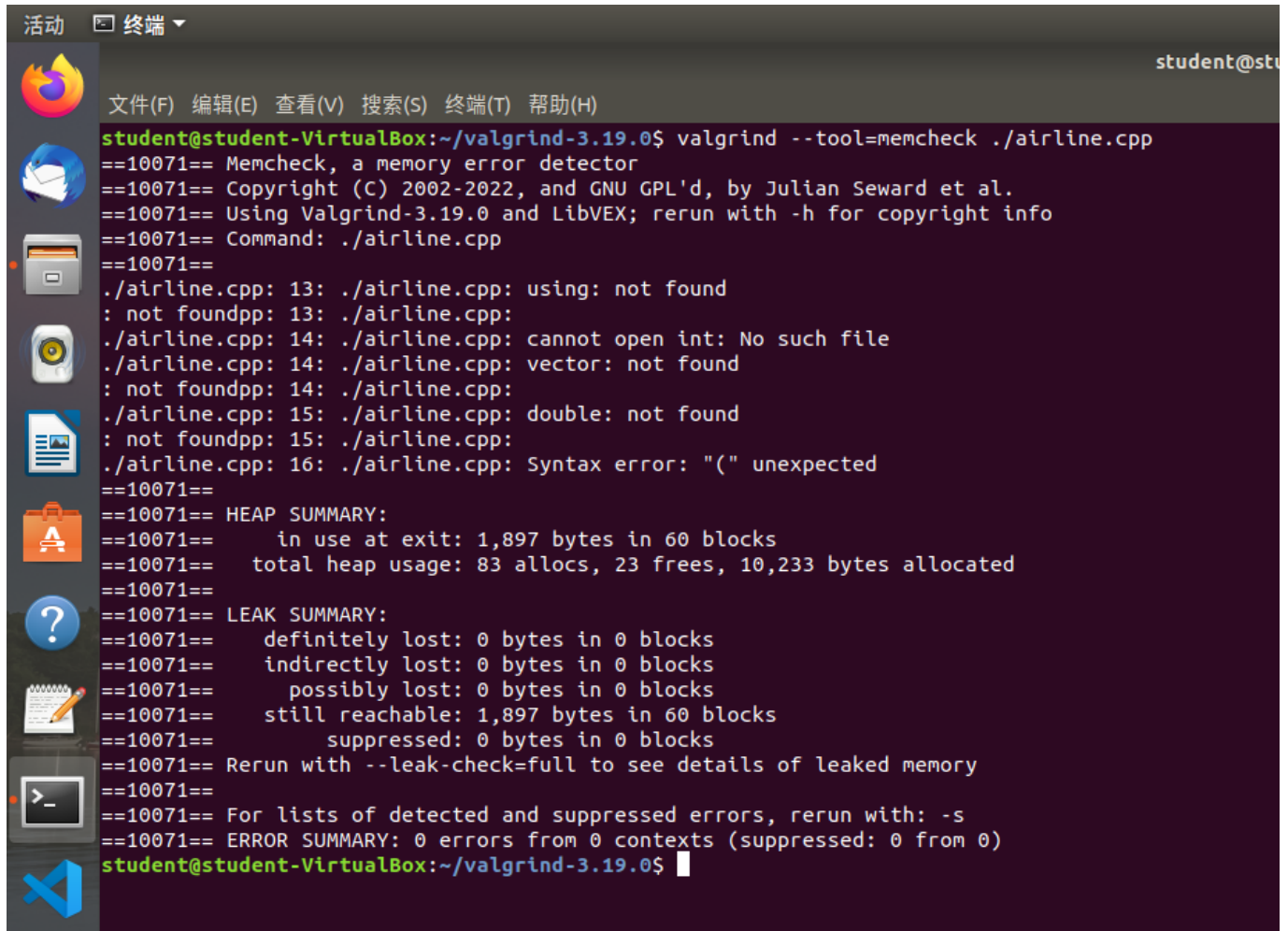


airline 实验报告

2021201645 赵宸

本次实验实现了全部功能，截止到写实验报告的今天，已经找助教检查完2.7及以前的全部功能，并准备在11.17日找助教检查2.8(届时如果有机会再更新实验报告).

内存泄漏检查



```
student@student-VirtualBox:~/valgrind-3.19.0$ valgrind --tool=memcheck ./airline.cpp
==10071== Memcheck, a memory error detector
==10071== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==10071== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==10071== Command: ./airline.cpp
==10071==
./airline.cpp: 13: ./airline.cpp: using: not found
: not foundpp: 13: ./airline.cpp:
./airline.cpp: 14: ./airline.cpp: cannot open int: No such file
./airline.cpp: 14: ./airline.cpp: vector: not found
: not foundpp: 14: ./airline.cpp:
./airline.cpp: 15: ./airline.cpp: double: not found
: not foundpp: 15: ./airline.cpp:
./airline.cpp: 16: ./airline.cpp: Syntax error: "(" unexpected
==10071==
==10071== HEAP SUMMARY:
==10071==    in use at exit: 1,897 bytes in 60 blocks
==10071==   total heap usage: 83 allocs, 23 frees, 10,233 bytes allocated
==10071==
==10071== LEAK SUMMARY:
==10071==    definitely lost: 0 bytes in 0 blocks
==10071==    indirectly lost: 0 bytes in 0 blocks
==10071==    possibly lost: 0 bytes in 0 blocks
==10071==    still reachable: 1,897 bytes in 60 blocks
==10071==    suppressed: 0 bytes in 0 blocks
==10071== Rerun with --leak-check=full to see details of leaked memory
==10071==
==10071== For lists of detected and suppressed errors, rerun with: -s
==10071== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
student@student-VirtualBox:~/valgrind-3.19.0$
```

由valgrind检查可知,本程序不存在内存泄漏现象。符合实验要求.

程序运行方法

2.1

输入:第一行输入1,表示运行第一个程序;第二行输入你要查询的机场.

输出:两行,第一行是用BFS得到的结果,第二行是用DFS得到的结果.

输入:1 <= n <= 79

输出:两行,每行均为1~79的全遍历.

2.2

输入:输入2表示要运行第二个程序,然后输入中转次数(-1表示无限制, 0表示直飞, 1表示一次中转, 2表示两次中转.....)

输出:N行, 一个N * N的01矩阵.

~~(样例放不下)~~

2.3

输入:输入3表示要运行第三个程序,然后输入两个机场的ID, 转机次数(<5);

输出:一个整数, 满足要求的全部航线数目.

样例:3 49 18 0

结果:4

2.4

输入:输入4表示运行第四个程序,然后输入两机场的ID.

输出:一个整数, 最短飞行时间(单位:分钟).

样例:4 39 10

结果:1315

2.5

输入:输入5 x表示运行2.5.x对应的程序;第二三四行依次输入:两个机场ID, 中转次数上限K;时段上限;时段下限.

输出:一行, 满足要求的航线(FlightID组成的顺序表), 不存在则输出-1.

若要运行2.5.3,则只需将时间段替换为飞机类型即可,其他不变.

样例:

```
5 1
28 74 4
5/5/2017 0:00
5/9/2017 0:00
```

结果:2113 403 1248 2251

2.6

要求同上.

2.7

输入:输入7表示要运行2.7,然后输入两个机场ID, 中转次数上限k, 中转时间上限m(分钟).

输出:满足要求的航线(FlightID组成的顺序表), 不存在则输出-1.

样例:7 39 10 4 610

结果:2300 283 377 1369

2.8

输入:输入8表示要运行2.8,然后输入两个机场ID, 中转时间上限m.

输出:第一行, 满足要求的航线(FlightID组成的顺序表), 不存在则输出-1.

第二行, 最低费用.不存在则无输出.

样例:8 9 68 600

结果:

```
1185 178 124 204 215 1229
6774
```

数据结构及算法

本程序采用了两种建图方式,第一种是完全二维的,以边为依据的建图,即课本中的建图方式,这种建图方式主要对应前四个小题;

第二种是以时间和位置双依据建图,同时在同位置不同时间之间搭建单向边.即首先根据每条航班的起点和终点建点,然后把该航班压入起始点中,同时在点之间建立连线. 空间复杂度: $O(n^2+n)=O(n^2)$. 时间复杂度: $O(n)$

功能的实现方式

2.1和2.2都是暴搜的板子题,不做赘述.

2.3的实现逻辑是先BFS暴搜每个节点的最小到达次数,然后再和输入的数据作比对,留做习题答案略,读者自证不难.

2.4

这个题一上来就有两个约束条件:遵循时序关系&时间最短.

如果不考虑时序关系,单纯追求时间最短,我们会怎么做呢?根据digkstra算法,每次把延伸到的顶点的所有边压栈,依据飞行时间最短排序;但如果考虑时序关系,那么我们就不能依据飞行时间排序了,而应该依据等待时间+飞行时间排序->那这不正是两次落地时间之差吗?而digkstra记录的是累计权值,也就正是上次落地时间+两次落地时间之差==最后一次的落地时间,那我们依据落地时间排序是不是就能得到最小的endtime了呢?

而且我们可以证明,中间任意时刻,落地时间更早都意味着我能“够到”更多的航班,因此局部最优也可以推广到全局最优.

据此,我们就得到了一个依据落地时间的魔改版digkstra算法,满足全局最优. 时间复杂度: $O(mn\log n)$,其中m为以出发点为起点的航班数.

2.5

从2.5开始,我更换了位置-时间的图,将原本的起点-终点图变成了起点-起飞时间-终点-落地时间图.

换图之后显然满足了时序关系.因此我直接对中转次数k做贪心即可.不加约束的时间复杂度为 $O(n\log n)$.

- 2.5.1可以仅在起点讨论起飞时的时间满足情况,不影响时间复杂度.
- 2.5.2我没有采用逆序建图的方式,而是直接在终点做筛选.这就意味着这已经不是一个正宗的digkstra算法了,但平均复杂度依然是 $O(n\log n)$,但最坏复杂度变成了 $O(n^2\log n)$,当且仅当结果为-1时取等.
- 2.5.3则每次压栈都检测这条航线是不是符合机型要求,不影响复杂度.

2.6

思路同上,只不过把中转次数换成航费即可,但与2.5相比少一个 $>k$ 就continue的剪枝.不做赘述.

2.7

这题又是一个双依据的dijkstra,为了严格卡dijkstra的定义,我采用了类似于WQS二分的思路,将中转次数和等待时间加权.权值依次渐进调整.这样就相当于在 $O(n \log n)$ 外边套了一层 $O(m)$,但显然,这个 m 我设的过大了,以至于这种做法比 $O(n^2 \log n)$ 还要慢.但我相信,如果图再大一点,我的加权的方式一定比 $O(n^2 \log n)$ 要快.当然,加权也有问题,由于我不知道等待时间和中转次数的取值范围,因此很难做归一化处理,所以我给二者做归一化的权值基本上等于是.....猜的.因此我担心,如果数据量增大之后,这种不正确的归一化会导致正确的权值从不正确的权值中间"漏过去"而导致错误.....当然只是猜想.

2.8

这次我深入的吸取了2.7的教训,没有采用加权的方式,而是类似2.5.2的剪枝思路,完全以航费为依据排序,然后减掉等待时间大于目标的枝子.....这种方法显然比2.7要快的多(sad).虽然最坏复杂度是 $O(n^2 \log n)$,但平均复杂度真的要低得多.

参考资料

- 1.数据结构(c++版) 邓俊辉
- 2.助教的群聊提示