物理综合EDA设计

1. 项目说明

1.1 项目介绍

- 实现选题一，平衡二分最小割算法
- 本组实现**FM算法**，在保证分割线两侧顶点数目相等的前提下，最小化分割的边数；分割过程中需要图数据结构和桶数据结构

1.1.1 组内分工

组内另外两位成员选择python实现划分，python实现部分的报告由另两位组员提交，这份报告是关于我尝试用C++实现整个流程，其完整流程包括

- 算法过程函数
- 图的底层类构造和输入文本读取
- 桶的底层类构造

1.1.2 编译环境

以下流程在系统版本为Ubuntu 24.04 LTS的Linux系统下，编译器以及相应工具链的版本如下的环境下可以成功运行。

- g++ (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
- gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
- clang version 14.0.0-1ubuntu1.1

1.1.3 运行

工作目录下直接运行`make`进行编译，得到可执行`division.out`

命令行格式如下，其作用分别为

- 直接运行，默认输入文件为`prob1.txt`，输出文件为`output.txt`，迭代次数为10
- 指定迭代次数并运行，输入、输出文件为默认
- 指定输入、输出文件并运行，迭代次数为默认
- 指定输入、输出文件和迭代次数并运行

```
<path>/division.out
<path>/division.out iteration_time
<path>/division.out input_file output_file
<path>/division.out input_file output_file iteration_time
```

2. FM算法

2.1 功能

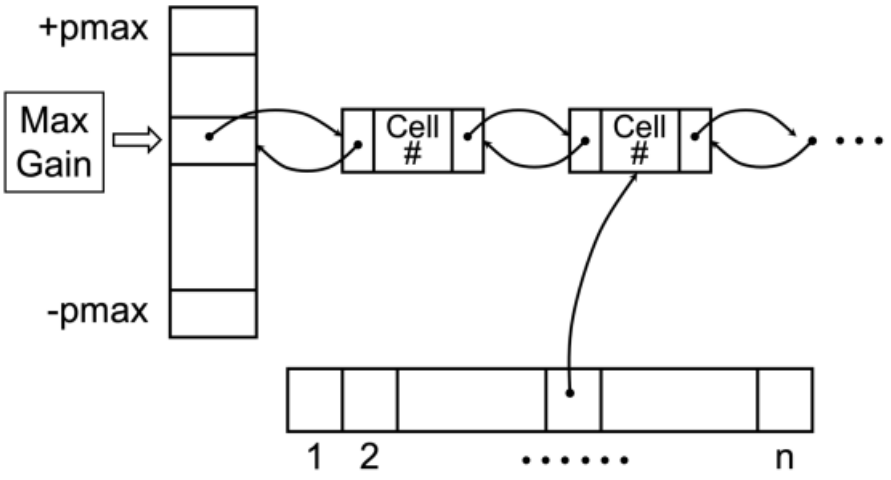
- 实现节点交换
- 完成节点交换后的增益更新（被交换的节点及其连接的节点）
- 比较并暂存累计增益，保留目前最优划分结果

使用**FM**算法来进行超图的划分，需要构造桶形结构来辅助我们寻找最大的增益节点并交换；

- 构造一个桶形结构，桶形结构的入口值对应于增益值，每个入口对应某个增益对应节点的链表，链表表元指向节点存储结构，
- 每个桶中的元素是节点指针数组，存储n个节点的指针，每个节点指针中保存了节点相关的信息
- 对每个节点可以通过桶形结构访问，也可以通过指针数组访问
- 可以从桶形结构找到增益最大入口，从相应链表中选择一个节点作为交换节点

- 节点被交换后，可以找到与该节点相连的节点，并通过指针数组直接访问这些节点更新增益，并在桶形结构中移动这些节点到对应增益入口链表中
- 通过这样的结构可以快速的更新增益，并快速查找到最大增益节点

算法的示意图如下所示：



2.2 实现细节

2.2.1 设计思路

从A桶移动到B桶

1. **从桶中选择增益最大点移出（基于bucket类的操作）**
    - 获得该节点增益`max_A / max_B`和节点序号`seq_A / seq_B`。
    - 同时从桶中删除节点。
2. **改变该移动节点的特性（基于graph类的操作）**
    - 根据得到的节点序号`seq_A / seq_B`在图中找到相应节点。
    - 更改该节点的域。
    - 改变该节点的增益（取反）。
3. **遍历图中该节点连接的点，改变增益并同时对桶进行重排序（基于bucket,graph类的操作）**
    - 遍历连接的点。
    - 如果连接的点和该点移动前同域：针对桶和图分别修改增益（+2）。
    - 如果连接的点和该点移动后同域：针对桶和图分别修改增益（-2）。

从B桶移动到A桶

与上述过程类似。

比较sum与sumMax，如果性能提升则暂存结果

sum:本轮交换之后的累计总增益。

sumMax:存放的历史最大的累计增益。

如果`sum>sumMax`：

- 更新sumMax;

- 暂存结果（记下此时的划分结果）

（如果此时增益没有相比最优增益提升，则不更新暂存的划分，最后只要取最后暂存的结果即是最优结果，这样就不用再运行前l次的移动）

输出文件

用`ofstream`实现将结果输出到文件`output.txt`。

循环修改

- 增设每次迭代的内部循环并调整函数放置位置。
- 写`result_store()`函数存放迭代后最优结果，作为下一次迭代的初始结果。

2.2.2 主要算法

(1) 桶形结构中单个节点的插入

- 函数：insert()
- 在拥有桶形结构后，我们需要为桶形结构加入insert函数进行单个节点的插入；
  - 对于还未有对应桶的增益节点，我们需要为其新增一个桶，并把节点插入作为首元素；
  - 对于已经有对应桶的增益节点，我们将其直接插在现成的桶的末尾；

```
// 插入结点V的指针，到对应的增益的list末尾
bool insert(int seq, Graph& G) {
        Vertex* V = G.getV(seq); // V[seq]的指针
        int gain = V->getGain(); // 查找V[seq]的增益
        list<Vertex*> vList;
        if (!getVlist(gain, vList)) {
            // 如果该增益值尚未在gainMap中创建
            list<Vertex*> vl;
            vl.push_back(V);
            gainMap[gain] = vl;
            return true;
        }
        gainMap[gain].push_back(V); // 插入到链表末尾
        return true;
    }
```

(2)现有的桶（或是已经改变的桶）的最大增益

- 函数：getMaxGain
- 由于每次交换节点时都需要找到最大的增益节点再进行交换，因而需要我们构造一个函数来得到现有的桶（或是已经改变的桶）的最大增益，由于图中的节点个数是确定的，因而我们可以为这个增益确定其阈值，从而遍历最大到最小直到找到非空的桶，即为最大增益值；

```
int getMaxGain(Graph& G) {
        int maxGain;
        int gain_uplimit = (G.getVnum() + 1) / 2;
        int gain_downlimit = -(G.getVnum() + 1) / 2;
        for (maxGain = gain_uplimit; maxGain >  gain_downlimit; maxGain--) {
            map<int, list<Vertex*>>::iterator it =  gainMap.find(maxGain);
            if (it == gainMap.end())
                continue;
            if (it->second.size() >= 1)
                return  maxGain;
        }
        return  false;
    }
```

(3) 桶中删除节点

- 函数：remove ()
- 由于在一次迭代过程中每次交换完节点后，被交换过的节点不允许再进行一次交换，从而我们相当于可以在桶中将其删除；
    - 需要注意的是，其在图中仍然存在，并且会影响与其想连接的节点的增益；
    - 因而删除操作只是会在桶中进行以降低交换和查找操作的复杂度；
    - 但在图中删除操作对应的是将节点的所属区域改变；

```
bool remove(int gain, int& seq) {
        if (gainMap[gain].size()!=0)
        {
            seq = gainMap[gain].front()->getSeq();
            gainMap[gain].pop_front();
            return true;
        }

        return false;
    }
```

(4) 遍历桶

- 函数：initial_bucketA，initial_bucketB
- 在每次迭代之前，我们都需要根据节点所属的区域将其插入到对应的桶中一遍后续交换和查找操作的进行；

```
void initial_bucketA(Graph* G)
    {

        for (Graph::VertexIterator iter = G->begin(G); iter != G->end(G);
iter++)
        {
            if (iter->getZone() == 0)
                insert(iter->getSeq(), *G);
        }
    }
    void initial_bucketB(Graph* G)
    {
```

```
        for (Graph::VertexIterator iter = G->begin(G); iter != G->end(G);
iter++)
        {
            if (iter->getZone() == 1)
                insert(iter->getSeq(), *G);
        }
    }
```

(5) 初始划分生成

- 函数：random_graph_zone
- 在进行所有的迭代之前我们需要将节点随机均匀划分到两块区域中，并计算他们的初始增益值；

```
void random_graph_zone(Graph* G) {
    int i = 0;
    for (Graph::VertexIterator iter = G->begin(G); iter != G->end(G);
iter++, i++) {
        if (i < G->getVnum() / 2) {
            iter->setZoneTmpStore(0);
            iter->setZone(0);
        }
        else {
            iter->setZoneTmpStore(1);
            iter->setZone(1);
        }
    }
    //给每个点设置初始增益
    for (Graph::VertexIterator iter = G->begin(G); iter != G->end(G);
iter++, i++) {
        for (Graph::EdgeInIterator iter_edge = G->begin(*iter); iter_edge !=
G->end(*iter); ++iter_edge) {
            if (iter_edge->getVout()->getZone() == iter->getZone()) iter-
>setGain(iter->getGain()+1);
            else iter->setGain(iter->getGain() - 1);;
        }

       // printf("%d\t", iter->getGain());
    }

}
```

(5) 记录交换最好结果

- 函数：temp_store
- 由于采用的FM算法是允许在判断有不好的增益（增益为负）出现时仍然进行交换，且要将交换进行到底的算法，所以需要将在交换过程中的最好结果记录下来，以供最后的输出使用；即图最后的输出结果为当增益累加到最大时的节点信息；

```
void temp_store(Graph* G) {
    for (Graph::VertexIterator iter = G->begin(G); iter != G->end(G); iter++)
    {
        iter->setGainTmpStore(iter->getGain());
        iter->setZoneTmpStore(iter->getZone());
    }
}
```

(6) 打印图

- 函数：print_vertex ()
- 在进行完一轮迭代之后我们需要将其值保存并作为下一轮迭代的开始指值；
    - 同时为了检验结果的正确性，我们将调整过后的图输出以检验其正确性；

```
void print_vertex(Graph* G, std::string const filename)
{
    ofstream fout;
    fout.open(filename);
    fout << "A ZONE :" << endl;
    for (Graph::VertexIterator iter = G->begin(G); iter != G->end(G);
iter++)
    {
        if (iter->getZone() == 0)
        {
            fout << iter->getSeq() << "   ";

        }
    }
    fout << endl;
    fout << "B ZONE :" << endl;
    for (Graph::VertexIterator iter = G->begin(G); iter != G->end(G);
iter++)
    {
        if (iter->getZone() == 1)
        {
            fout << iter->getSeq() << "   ";

        }
    }
    fout.close();
}
```

2.3 图数据结构设计

2.3.1 图

本实验中基本类包含下述class；图中使用改进的线性表存储Vertex和Edge对象，使用十字链表法存储Vertex和Edge之间的相互关系，并且提供迭代器方便进行Vertex或Edge的遍历。

```
class Graph;                        // 图类
class Graph::VertexIterator;        // 迭代器 遍历图中所有的顶点
class Graph::EdgeIterator;          // 迭代器 遍历图中所有的边
class Graph::EdgeInIterator;        // 迭代器 遍历指定顶点的所有指向它的边
class Graph::EdgeOutIterator;       // 迭代器 遍历指定顶点的所有它指向的边
class Edge;                         // 边类
class Vertex;                       // 顶点类
```

1.线性表数据的存储

对于一般的线性表存储方式，有以下特点

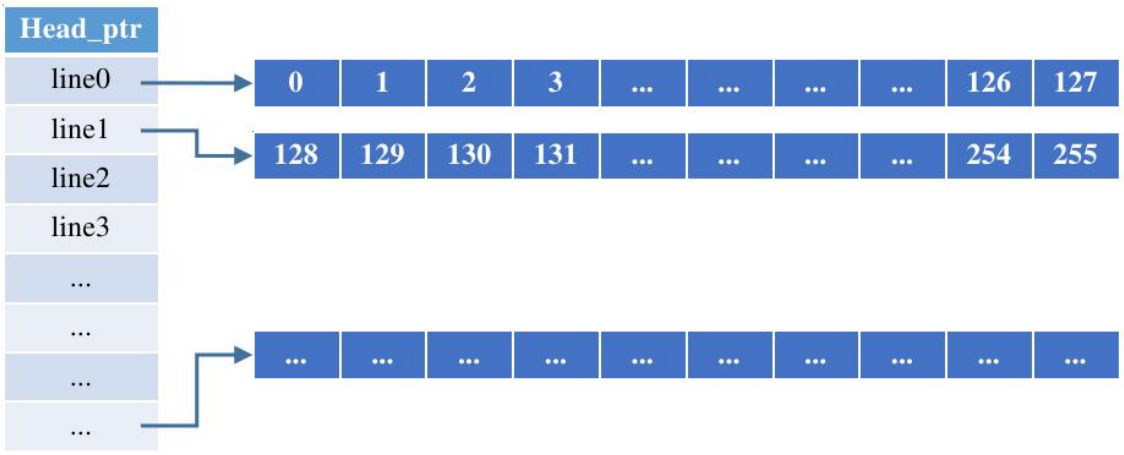
- 顺序表（数组或vector）直接存储对象、索引方便，但是扩大容量时需要重新分配另一块内存，导致原先指向元素的指针全部失效，导致严重的功能错误
- 链表（单链表或双链表）需要存储指向前后件的指针，有冗余的存储空间，且索引不方便；但元素加入后不会改变其位置，不会导致指针失效

此处使用一个改进结构，该结构类似二级数组，如下图所示

- 内级是固定大小的数组，且每个内级数组的大小都相同；内级数组支持运行时动态分配，一经分配后位置和大小不会改变
- 外级是存储数组头指针的vector；当vector存满时，会重新分配一块内存，但只重新移动数组首指针，不会移动数组元素

这种结构需要的冗余存储空间较少，完全解决了数组元素位置移动的问题；并且每次分配和析构一大块内存，比逐个元素分配更高效；可以直接索引访问，但是需要进行映射转换，对索引方法封装提出了更高的要求

在`Graph`类中使用这种线性表存储`Vertex`和`Edge`对象，`Vertex`对象允许索引访问，`Edge`对象只允许通过入点和出点指针`Vertex*`进行访问；提供迭代器遍历`Vertex`和`Edge`对象，后续小节中会进行说明

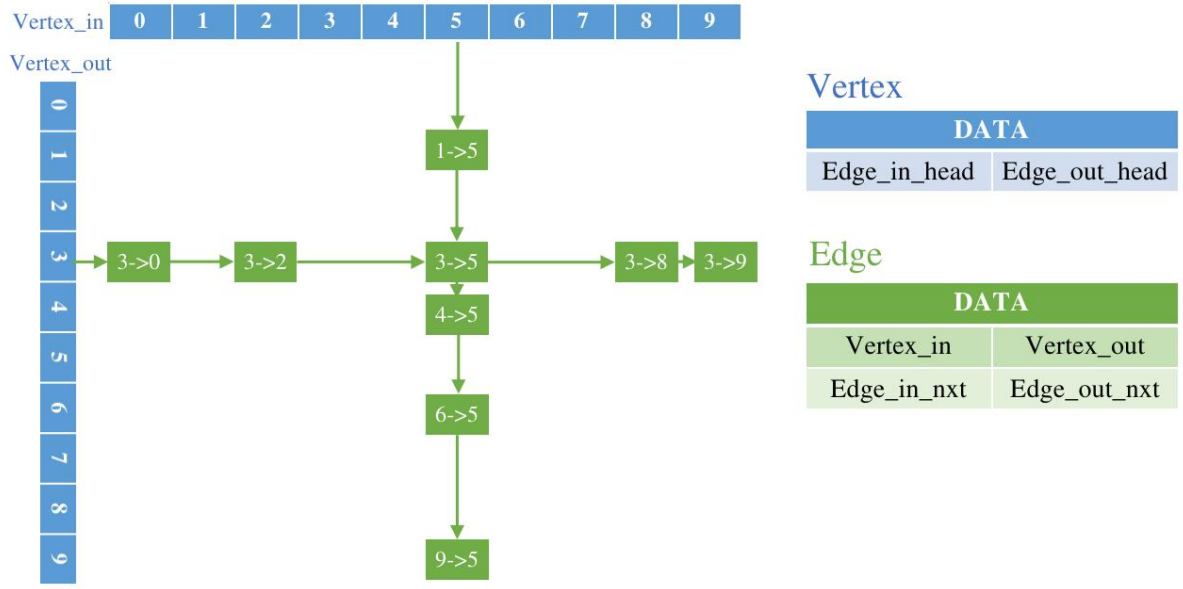


2. 数据相互关系的存储

由于$|E| << |V|^2$，该图是一个稀疏图；图中顶点和边的互连关系采用十字链表法存储

- 顶点类存储所有"指向该顶点的边"的首指针、所有"从该顶点出发的边"的首指针；数据域中存储入度、出度、顶点所属的划分区域、顶点的增益值、在累计增益最大值的情况下顶点所属的划分区域、在累计增益最大值的情况下顶点的增益值，同时也存储该顶点的索引值
- 边类存储入点的指针、出点的指针、下一条具有相同入点的边指针、下一条具有相同出点的边指针

- 从一个点出发，可以以单链表的方式访问所有指向该顶点的边，也可以以单链表的方式访问从该顶点出发的边



3. 图类的迭代器

对图类中的对象，提供四个迭代器类以便于遍历；这四个类嵌套定义在`Graph`类中

```
class Graph::VertexIterator;        // 迭代器 遍历图中所有的顶点
class Graph::EdgeIterator;          // 迭代器 遍历图中所有的边
class Graph::EdgeInIterator;        // 迭代器 遍历指定顶点的所有指向它的边
class Graph::EdgeOutIterator;       // 迭代器 遍历指定顶点的所有它指向的边
```

使用迭代器可以很方便地遍历范围内的所有对象；迭代器最基础的使用模式如下

```
for(Graph::xxxIterator iter=g.begin(...); iter=g.end(...); ++iter)
```

迭代器使用时几乎等价于元素指针，其具体使用模式如下

其中`begin()`函数和`end()`函数的入参决定返回迭代器的类型，具体要求见下一小节

```
iter=g.begin(...);      // 获取某个图对象中该迭代器的起点
iter=g.end(...);        // 获取某个图对象中该迭代器的终点
iter=iter2;             // 将一个迭代器赋值给另一个迭代器
iter==iter2;            // 比较两个迭代器是否相等
iter!=iter2;            // 比较两个迭代器是否不相等
iter++; ++iter;         // 迭代器移动到下一个位置
iter->...               // 指向当前访问的元素的类成员 用法相当于元素指针
(*iter)                 // 当前访问的元素的引用 用法相当于元素指针
```

4. 类的函数接口

`Graph, Vertex, Edge`三个类的接口如下所示

5. 根据给定文本信息构造图

- 思路：
  1. 实例化一个Graph结构。`Graph* g=new Graph;`
  2. 接着调用`fstream`标准库里的`ifstream`类型表示输入文件流，用于从`prob1.txt`读取信息。

```
    ifstream fin;
    fin.open("prob1.txt", ios::in);
    if (!fin.is_open())
    {
        cout << "无法找到这个文件！" << endl;
        return 1;
    }
```

  3. 调用`getline()`，一行一行地读文件到缓冲区`char buffer[256]`

```
fin.getline(buffer, 100);
```

4. txt文件的第一行是节点总数， 后面所有行对应着节点的指向关系（以有向图处理）。第二行开始每一行的第一个数据是该节点的编号，第二个数据开始是其指向的节点的编号。因此为了精确获取每一个节点的连接，在读文件过程中用变量`count_tmp`记录行数，用变量`countinV`记录当前数据的位置。
5. 读取第一行时，在图中创建“节点数目”个节点类型

```
if (count_tmp == 0) {
                for (int i = 0; i < p; i++) {
                    Vertex* v = g->getV(i);
                }
            }
```

6. 调用标准库`<sstream>`中的`stringstream`类型，用以将缓存区中的`char`转为`string`，再从`string`转为`int`

```
ss << buffer;
ss >> p;
```

7. 用变量`nowv`记录当前行的节点编号。当当前数据p是本行第一个时，意味着这一行后面的节点（编号）都与p相连。在图中构造边即可。

```
if (countinV == 0) {
                    nowv = p;
                }
                else {
                    Edge* e= g->getE(g->getV(p), g->getV(nowv));
                }
```

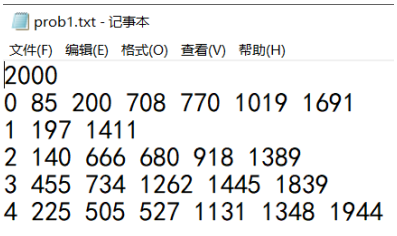
8. 读到文件末尾`eof`时循环结束，关闭文件流

```
    while (!fin.eof()) {}
    ...
    fin.close();
```

- 结果打印：调用Edge迭代器打印整个图的所有边

```
    for(Graph::EdgeIterator iter=g->begin(*g); iter!=g->end(*g); ++iter){
        cout<<iter->getVout()->getSeq()<<"->"<<iter->getVin()->getSeq()<<endl;
    }
```

原txt文件



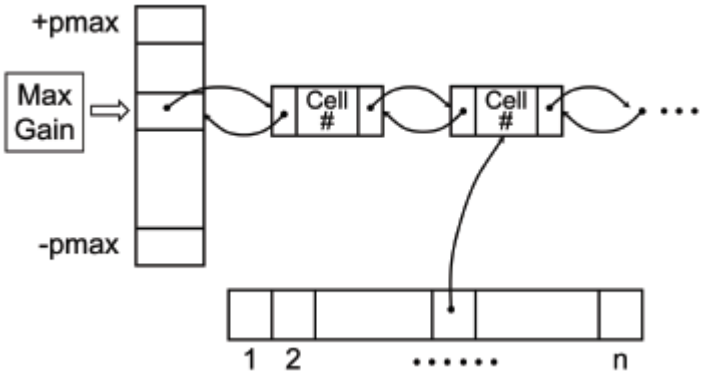
图中的边连接关系：

```
0->85
0->200
0->708
0->770
0->1019
0->1691
1->197
1->1411
2->140
2->666
2->680
2->918
2->1389
3->455
3->734
3->1262
3->1445
3->1839
4->225
4->505
4->527
4->1131
4->1348
4->1944
```

2.3.2 基本类-桶

FM算法的每一轮迭代将节点集均匀分配到两个桶形结构中。每次交换，从一个桶形结构中取增益最大的节点，交换到另一个节点集合中，从而避免了KL算法查询交换后增益最大的节点对导致的高复杂度，将整体复杂度限制在$O(E)$。

利用gainmap（map<int,list>）来表示这一桶形数据结构，其中的int表示对应的增益，list表示增益为int的节点构成的双向链表。FM算法要求每次从划分X中找到最大增益的节点A，并将其加入X'中，之后从X'中找到最大增益的节点B并将其加入X。这之中涉及到了从划分中找到最大增益节点、从划分X中删掉最大增益的节点、在划分X中加入节点并更新增益。



桶形结构要求用增益数值作为入口，以增益值对应节点链表。本实验中使用C++标准模板库的关联容器map，将增益数值gain与节点链表直观地进行关联。节点链表使用标准模板库的双向链表list存储节点指针Vertex*。

```
class Vertex; // 节点类
class Bucket{ // 桶类
private:
    map<int, list<Vertex*>> gainMap;
}
```

1. 主要算法

(1) getVlist:

- 通过增益值查询节点链表，返回节点链表引用。

(2) insert:

- 将编号为seq的节点Vertex放入桶形结构，插入的位置是增益list的末尾。

(3) changeGain:

- 对位于桶形结构中的节点，更改其增益；将节点序号为seq的点的增益改为newGain，并从gainmap原来的list中将其删掉、加入修改后的list中。

(4) getMaxGain:

- 查询当前桶形结构中的最大增益值；从最大可能的增益开始，不断向下搜索，直到所查找的增益有对应的list为止。

(5) remove:

- 删除增益值p对应的节点链表的第一个节点，并返回其节点编号；这里我们选择删除的节点是双向链表的最后一项。

实验结果

命令行打印如下

```
═══════════ Guidance ═══════════
<path>/division.out
        Running division process with default input file "prob1.txt" and output file "output.txt", and ite
ration time is 10.
<path>/division.out iteration_times
        Running division process with manually set and iteration times, and default input file "prob1.txt"
 and output file "output.txt"
<path>/division.out input_file_path output_file_path
        Running division process with manually set input file and output file. The iteration time is 10.
<path>/division.out input_file_path output_file_path iteration_times
        Running division process with manually set input file, output file and iteration time.
═══════════ Processing ═══════════
Before iteration: 5110 edges are cut.1000 vertexes in each zone.
Iteration 1: 2152 edges are cut. 1000 vertexes in each zone.
Iteration 2: 1984 edges are cut. 1000 vertexes in each zone.
Iteration 3: 1982 edges are cut. 1000 vertexes in each zone.
Iteration 4: 1978 edges are cut. 1000 vertexes in each zone.
Iteration 5: 1970 edges are cut. 1000 vertexes in each zone.
Iteration 6: 1964 edges are cut. 1000 vertexes in each zone.
Iteration 7: 1964 edges are cut. 1000 vertexes in each zone.
Iteration 8: 1964 edges are cut. 1000 vertexes in each zone.
Iteration 9: 1964 edges are cut. 1000 vertexes in each zone.
Iteration 10: 1964 edges are cut. 1000 vertexes in each zone.
```

- 程序运行时，首先打印命令行使用格式，然后进行迭代分割；每次迭代都会输出当前的切割边数和两侧的顶点数
- 运行完毕后，划分情况被输出到`output.txt`中

```
A ZONE :
```

2

1 2 4 5 6 7 8 9 10 12 14 15 16 17 18 19
22 23 24 27 29 31 36 40 41 42 44 46 51 52 53
54 56 58 60 61 62 63 64 66 68 70 71 72 73
75 79 81 83 87 90 91 93 95 96 99 101 103 106
107 108 109 110 113 114 115 118 120 122 124 125
131 133 140 142 147 150 151 153 154 156 157 158
159 160 161 163 164 171 174 175 178 180 181 182
185 186 187 189 190 193 194 197 199 203 206 208
209 211 212 213 214 215 216 217 218 220 222 225
229 230 232 234 235 236 242 243 245 246 247 248
250 251 252 253 255 256 258 259 260 262 264 265
267 270 272 274 278 280 281 282 283 285 286 287
288 290 291 294 298 302 303 304 307 308 314 315
316 320 321 323 325 326 327 328 333 335 337 339
340 342 346 348 349 350 351 352 353 354 356 357
358 359 362 364 365 367 369 370 371 372 373 375
377 379 381 385 389 392 393 394 395 397 398 399
401 404 405 409 416 417 418 419 422 423 424 425
426 427 428 429 430 433 435 437 439 440 441 442
450 451 452 453 454 456 460 463 467 473 479 481
483 484 485 487 488 489 490 492 495 498 500 501
502 503 505 507 508 510 511 512 513 516 517 525
526 527 528 530 531 534 536 537 538 543 545 548
549 552 555 556 558 561 563 564 566 570 571 572
573 574 576 578 579 583 586 587 588 590 591 595
596 598 599 600 602 607 610 611 612 614 615 624
625 626 627 635 636 637 638 640 641 642 643 646
648 649 650 652 656 657 658 661 663 665 666 667
668 670 671 676 677 678 681 682 683 685 688 689
692 694 695 697 698 699 700 701 702 703 706 707
709 711 713 715 717 722 724 726 727 728 729 732
733 734 735 736 738 739 740 741 742 743 745 746
748 751 752 753 754 755 758 759 761 762 764 765
766 767 768 771 773 775 776 780 781 782 784 785
786 789 790 791 792 794 795 798 799 801 803 805
807 808 811 812 817 818 820 821 822 824 825 830
831 836 837 840 841 843 845 846 849 851 853 854
857 858 860 861 866 868 870 871 874 875 877 879
880 882 883 884 886 888 889 891 892 893 895 896
897 901 902 905 910 911 917 918 919 920 924 929
932 936 937 938 939 941 942 943 946 948 950 951
953 954 957 958 959 961 962 966 967 969 972 973
974 975 976 977 980 981 982 983 987 988 989 991
992 994 999 1001 1003 1005 1006 1009 1011 1013 1016
1018 1020 1021 1022 1026 1027 1031 1036 1038 1039
1040 1042 1045 1046 1048 1051 1053 1056 1057 1058
1059 1061 1062 1064 1071 1074 1076 1079 1080 1081
1082 1083 1086 1087 1091 1093 1095 1096 1097 1100
1102 1103 1105 1106 1109 1110 1118 1119 1125 1127
1131 1133 1134 1135 1136 1137 1138 1139 1140 1141
1142 1151 1152 1153 1156 1157 1158 1159 1160 1165
1167 1168 1169 1172 1173 1176 1181 1184 1186 1187
1188 1189 1190 1192 1193 1194 1197 1198 1200 1202
1203 1204 1210 1211 1212 1213 1214 1215 1216 1218
1219 1220 1221 1223 1226 1228 1229 1232 1233 1234
1235 1236 1238 1239 1241 1242 1245 1246 1248 1249

1254 1256 1260 1261 1265 1269 1270 1271 1272 1273
1274 1275 1276 1284 1285 1286 1287 1289 1292 1295
1296 1297 1300 1303 1304 1305 1306 1307 1308 1310
1313 1329 1330 1334 1335 1336 1337 1339 1340 1343
1346 1347 1348 1350 1351 1353 1354 1355 1358 1359
1360 1362 1366 1368 1370 1371 1372 1375 1377 1378
1379 1387 1388 1389 1394 1397 1399 1400 1407 1408
1410 1411 1412 1414 1416 1417 1418 1420 1421 1425
1427 1431 1432 1435 1443 1444 1446 1447 1449 1452
1454 1455 1458 1459 1462 1463 1465 1467 1468 1469
1473 1476 1480 1483 1484 1491 1493 1499 1500 1504
1508 1513 1516 1517 1521 1522 1523 1524 1525 1526
1529 1530 1531 1534 1535 1537 1538 1540 1541 1542
1543 1544 1550 1555 1556 1557 1558 1561 1563 1565
1566 1567 1568 1572 1573 1575 1579 1582 1583 1585
1588 1589 1592 1594 1595 1604 1605 1606 1610 1611
1614 1615 1616 1617 1620 1621 1622 1623 1627 1630
1634 1635 1638 1641 1642 1645 1646 1647 1649 1650
1651 1653 1661 1662 1663 1664 1668 1669 1672 1674
1675 1676 1680 1688 1695 1696 1697 1699 1700 1707
1708 1709 1710 1711 1712 1713 1717 1720 1722 1724
1725 1726 1728 1729 1730 1732 1733 1735 1736 1737
1740 1742 1744 1745 1747 1751 1752 1753 1754 1755
1756 1757 1758 1759 1760 1761 1766 1767 1769 1770
1772 1774 1779 1781 1782 1785 1789 1790 1797 1799
1802 1808 1812 1813 1816 1817 1822 1823 1827 1828
1829 1830 1832 1834 1837 1839 1843 1844 1847 1850
1852 1853 1854 1857 1858 1859 1861 1864 1869 1870
1872 1873 1876 1877 1878 1880 1881 1882 1883 1884
1885 1888 1890 1891 1900 1901 1902 1904 1905 1906
1908 1909 1910 1911 1913 1916 1917 1919 1920 1932
1934 1935 1936 1937 1940 1941 1942 1944 1947 1952
1953 1955 1956 1960 1961 1965 1969 1970 1971 1975
1978 1979 1980 1982 1987 1989 1990 1991 1995 1996
1998 1999

B ZONE :

4

0 3 11 13 20 21 25 26 28 30 32 33 34 35 37
38 39 43 45 47 48 49 50 55 57 59 65 67 69 74
76 77 78 80 82 84 85 86 88 89 92 94 97 98
100 102 104 105 111 112 116 117 119 121 123 126
127 128 129 130 132 134 135 136 137 138 139 141
143 144 145 146 148 149 152 155 162 165 166 167
168 169 170 172 173 176 177 179 183 184 188 191
192 195 196 198 200 201 202 204 205 207 210 219
221 223 224 226 227 228 231 233 237 238 239 240
241 244 249 254 257 261 263 266 268 269 271 273
275 276 277 279 284 289 292 293 295 296 297 299
300 301 305 306 309 310 311 312 313 317 318 319
322 324 329 330 331 332 334 336 338 341 343 344
345 347 355 360 361 363 366 368 374 376 378 380
382 383 384 386 387 388 390 391 396 400 402 403
406 407 408 410 411 412 413 414 415 420 421 431
432 434 436 438 443 444 445 446 447 448 449 455
457 458 459 461 462 464 465 466 468 469 470 471
472 474 475 476 477 478 480 482 486 491 493 494
496 497 499 504 506 509 514 515 518 519 520 521
522 523 524 529 532 533 535 539 540 541 542 544
546 547 550 551 553 554 557 559 560 562 565 567
568 569 575 577 580 581 582 584 585 589 592 593
594 597 601 603 604 605 606 608 609 613 616 617
618 619 620 621 622 623 628 629 630 631 632 633
634 639 644 645 647 651 653 654 655 659 660 662
664 669 672 673 674 675 679 680 684 686 687 690
691 693 696 704 705 708 710 712 714 716 718 719
720 721 723 725 730 731 737 744 747 749 750 756
757 760 763 769 770 772 774 777 778 779 783 787
788 793 796 797 800 802 804 806 809 810 813 814
815 816 819 823 826 827 828 829 832 833 834 835
838 839 842 844 847 848 850 852 855 856 859 862
863 864 865 867 869 872 873 876 878 881 885 887
890 894 898 899 900 903 904 906 907 908 909 912
913 914 915 916 921 922 923 925 926 927 928 930
931 933 934 935 940 944 945 947 949 952 955 956
960 963 964 965 968 970 971 978 979 984 985 986
990 993 995 996 997 998 1000 1002 1004 1007 1008
1010 1012 1014 1015 1017 1019 1023 1024 1025 1028
1029 1030 1032 1033 1034 1035 1037 1041 1043 1044
1047 1049 1050 1052 1054 1055 1060 1063 1065 1066
1067 1068 1069 1070 1072 1073 1075 1077 1078 1084
1085 1088 1089 1090 1092 1094 1098 1099 1101 1104
1107 1108 1111 1112 1113 1114 1115 1116 1117 1120
1121 1122 1123 1124 1126 1128 1129 1130 1132 1143
1144 1145 1146 1147 1148 1149 1150 1154 1155 1161
1162 1163 1164 1166 1170 1171 1174 1175 1177 1178
1179 1180 1182 1183 1185 1191 1195 1196 1199 1201
1205 1206 1207 1208 1209 1217 1222 1224 1225 1227
1230 1231 1237 1240 1243 1244 1247 1250 1251 1252
1253 1255 1257 1258 1259 1262 1263 1264 1266 1267
1268 1277 1278 1279 1280 1281 1282 1283 1288 1290
1291 1293 1294 1298 1299 1301 1302 1309 1311 1312
1314 1315 1316 1317 1318 1319 1320 1321 1322 1323
1324 1325 1326 1327 1328 1331 1332 1333 1338 1341

```
1342  1344  1345  1349  1352  1356  1357  1361  1363  1364
1365  1367  1369  1373  1374  1376  1380  1381  1382  1383
1384  1385  1386  1390  1391  1392  1393  1395  1396  1398
1401  1402  1403  1404  1405  1406  1409  1413  1415  1419
1422  1423  1424  1426  1428  1429  1430  1433  1434  1436
1437  1438  1439  1440  1441  1442  1445  1448  1450  1451
1453  1456  1457  1460  1461  1464  1466  1470  1471  1472
1474  1475  1477  1478  1479  1481  1482  1485  1486  1487
1488  1489  1490  1492  1494  1495  1496  1497  1498  1501
1502  1503  1505  1506  1507  1509  1510  1511  1512  1514
1515  1518  1519  1520  1527  1528  1532  1533  1536  1539
1545  1546  1547  1548  1549  1551  1552  1553  1554  1559
1560  1562  1564  1569  1570  1571  1574  1576  1577  1578
1580  1581  1584  1586  1587  1590  1591  1593  1596  1597
1598  1599  1600  1601  1602  1603  1607  1608  1609  1612
1613  1618  1619  1624  1625  1626  1628  1629  1631  1632
1633  1636  1637  1639  1640  1643  1644  1648  1652  1654
1655  1656  1657  1658  1659  1660  1665  1666  1667  1670
1671  1673  1677  1678  1679  1681  1682  1683  1684  1685
1686  1687  1689  1690  1691  1692  1693  1694  1698  1701
1702  1703  1704  1705  1706  1714  1715  1716  1718  1719
1721  1723  1727  1731  1734  1738  1739  1741  1743  1746
1748  1749  1750  1762  1763  1764  1765  1768  1771  1773
1775  1776  1777  1778  1780  1783  1784  1786  1787  1788
1791  1792  1793  1794  1795  1796  1798  1800  1801  1803
1804  1805  1806  1807  1809  1810  1811  1814  1815  1818
1819  1820  1821  1824  1825  1826  1831  1833  1835  1836
1838  1840  1841  1842  1845  1846  1848  1849  1851  1855
1856  1860  1862  1863  1865  1866  1867  1868  1871  1874
1875  1879  1886  1887  1889  1892  1893  1894  1895  1896
1897  1898  1899  1903  1907  1912  1914  1915  1918  1921
1922  1923  1924  1925  1926  1927  1928  1929  1930  1931
1933  1938  1939  1943  1945  1946  1948  1949  1950  1951
1954  1957  1958  1959  1962  1963  1964  1966  1967  1968
1972  1973  1974  1976  1977  1981  1983  1984  1985  1986
1988  1992  1993  1994  1997
```

- **程序运行正确，完成了顶点数相等的二分割**
- **程序运行效果明显，最终结果相比于初始解的切割边数有明显的下降，约降至38%**；每次全图迭代的边际效应递减