第一部分：文件读取

在文件读取过程中，为了实现较高的读取效率，程序采用 fread 函数，将文件数据一次性读取到内存空间当中，然后再对数据进行解析。

```
FILE* stream = NULL;
uint32_t char_num, len;
char* p = NULL;
stream = freopen(filename, "r", stdin);
if (stream)
{
    fseek(stream, 0, SEEK_END);
    char_num = ftell(stream);
    fseek(stream, 0, SEEK_SET);

    p = new char[char_num];
    len = fread(p, 1, char_num, stdin);

    data_analyse(p, len, id3_map, data, mod);
    delete[] p;
    p = NULL;
    fclose(stream);
    return true;
}
```

第二部分：数据解析

通过分析数据文件我们可以知道，文件中每一条记录以'\n'区分，即每条记录单独成一行，每条记录有三个数据分别是 id1、id2、id3，并且数据之间以','分隔。因此在数据解析时，我根据这一特点来设计了 data_analyse 函数来实现数据解析功能。

当我们从内存中读取到','时，就表示已经获取了一个完整的数据，然后将这个数据转换为 int 类型，并存储到临时容器中，当从内存中读取到'\n'时，就表示已经读取完了一条记录了，此时将临时容器中的数据存储到指定的容器中。

（这部分代码较长，不方便贴出，可在源码中查看。）

第三部分：join 方法

通过解析题目语句我们可以发现，题目要求我们将所有 id3 相同的记录提取出来，因此我在解析第一个文件数据的同时，建立了一个 unordered_map，map 的 key 值为 Id3，value 为一个容器，用来存放所有 Id3 = key 值的记录。

由于我是先解析文件 1、再解析文件 2，因此在解析文件 2 时，unordered_map 已经建立完成，此时我们根据 Id3 在 unordered_map 中进行查找，如果能找到记录，则说明这个 Id3 是文件 1、文件 2 共有的，此时将记录从 unordered_map 中提取出来并与文件 2 的该条记录合并，然后存放到指定数组中，否则不做操作。

由于接下来的操作与 Id3 无关，因此在数据合并时我们只需合并两个文件的 id1、id2 数据即可，而将 Id3 记录丢弃。

（该方法在 data_analyse 函数中）

```
if (mod == 1) //处在解析文件1模式
{
    if (id3_map.find(res) == id3_map.end()) //表示没有加入过
    {
        id3_map.insert(pair<int, vector<vector<int>>>(res, temp));
        id3_map[res].emplace_back(lineArray);
    }
    else
        id3_map[res].emplace_back(lineArray);
}
else if (mod == 2)  //处在解析文件2模式
{
    if (id3_map.find(res) != id3_map.end()) //表示id3有值
    {
        for (auto& x : id3_map[res])
        {
            mergearr[0] = x[0];             //文件1的 id1
            mergearr[1] = x[1];             //文件1的 id2
            mergearr[2] = lineArray[0];     //文件2的 id1
            mergearr[3] = lineArray[1];     //文件2的 id2
            data.emplace_back(mergearr);    //将合并后的记录转存到data中
        }
    }
}
```

第四部分：group 方法

题目要求，将数据根据 t1.id2、t2.id2 进行分组，因此我首先对合并后的数据根据 t1.id2、t2.id2 进行升序排序，然后采用 for 循环遍历所有数据。

在遍历数据的过程中，我检测当前数据的 t1.id2、t2.id2 与前一条数据的 t1.id2、t2.id2 是否相等，如果相等，则表示这两条数据处在同一组，同时，我还创建了两个变量 max_t1_id1、min_t2_id1 分别来记录同一组中 t1.id1 的最大值以及 t2.id1 的最小值。

当检测到当前数据的 t1.id2、t2.id2 与前一条数据的 t1.id2、t2.id2 不相等时，则表示这两条数据不在同一组，此时我会将上一组的信息（max_t1_id1、min_t2_id1、t1.id2、t2.id2）转存至另一个容器中，然后重新对 max_t1_id1、min_t2_id1 进行赋值。

（该方法在 group 函数中）

```
sort(j_arr.begin(), j_arr.end(), [](vector<int>& fir, vector<int>& sec)
    {
        if (fir[1] == sec[1])           //根据t1.id2、 t2.id2排序
            return fir[3] < sec[3];
        else
            return fir[1] < sec[1];
    });
```

```
for (i = 1; i < size; i++)
{
    if ((j_arr[i - 1][1] == j_arr[i][1]) && (j_arr[i - 1][3] == j_arr[i][3]))
    {
        j_arr[i][0] > max_t1_id1 ? max_t1_id1 = j_arr[i][0] : NULL;
        j_arr[i][2] < min_t2_id1 ? min_t2_id1 = j_arr[i][2] : NULL;
    }
    else
    {
        ans[cnt][0] = max_t1_id1;
        ans[cnt][1] = min_t2_id1;
        ans[cnt][2] = j_arr[i - 1][1];
        ans[cnt][3] = j_arr[i - 1][3];
        max_t1_id1 = j_arr[i][0];
        min_t2_id1 = j_arr[i][2];
        cnt++;
    }
}
ans[cnt][0] = max_t1_id1;
ans[cnt][1] = min_t2_id1;
ans[cnt][2] = j_arr[i - 1][1];
ans[cnt][3] = j_arr[i - 1][3];
cnt++;
```

第五部分：order by 方法

题目要求根据 max(t1.id1)、t2.id2、t1.id2 进行升序排序，由于经过第四部分的数据整理后，每条数据都记录一个组的 max(t1.id1)、min(t2.id1)、以及组号(t1.id2、t2.id2)，因此我仅需对第四部分整理的数据按照 max(t1.id1)、t2.id2、t1.id2 进行升序排序即可实现题目要求。

（该方法在 group 函数中）

```
sort(ans.begin(), ans.end(), [](vector<int>& fir, vector<int>& sec)
    {
        if (fir[0] == sec[0])
        {
            if (fir[3] == sec[3])
                return fir[2] < sec[2];
            else
                return fir[3] < sec[3];
        }
        else
            return fir[0] < sec[0];
    });
```

第六部分：select 方法

由于我的 ans 容器存放的是每组的 max(t1.id1)、min(t2.id1)、以及组号(t1.id2、t2.id2)，并且 ans 已再第五部分按照题目的要求进行了排序，因此我们仅需按顺序遍历 ans 容器，并输出每个子容器的前两个元素即可满足题目要求。

（该方法在 main 函数中）

```
for (auto& x : ans)
    printf("%d,%d\n", x[0], x[1]);
```