CNNS Class Project SMPC 报告

Part 1. Basic Theory

```
You should select two of the following techniques: (secret sharing, oblivious transfer, homomorphic encryption, and garbled circuits) and give a biref introduction on them. You should also pay extra attention on their Cryptograghy rationales. (20%)
```

Part 1a: a biref introduction on two of the following techniques

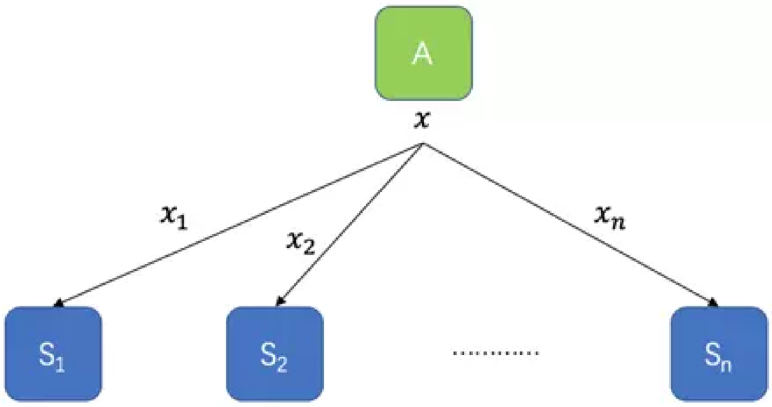
第 1a 部分：对所要求的技术四选二进行介绍

1. secret sharing

资料来自知乎——密钥分享

- cryptograghy rationales（密码学基本原理）

```
basic
```

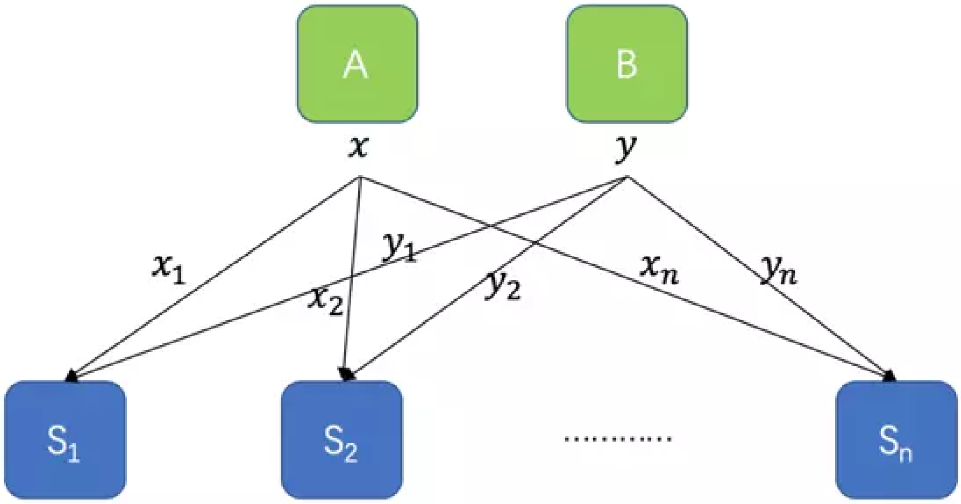


那密钥分享具体是怎么运作的呢？我们先从一个最简单的方法讲起。假设A这个人有一个秘密数字 x ，他想将其分发到$S_1, S_2, ..., S_n$那里。那么A首先要做的便是生成 $n-1$ 个随机数 $r_1, r_2, ..., r_n$ ，然后计算第 n 个数 $r_n = x - \sum_{i=1}^{n-1} r_i$ ，最后A令 $x_1 = r_1, x_2 = r_2, ..., x_n = r_n$ ，并将它们发给$S_1, S_2, ..., S_n$。上面这种简单的方法具有如下几条性质：

1. 各个数字 $x_1, x_2, ..., x_n$ 都是随机分布的，单独一个或若干个并不泄露任何信息；

2. 当所有$x_1, x_2, ..., x_n$合在一起时，可以还原 x ，因为 $x = \sum_{i=1}^{n} x_i$ ；

3. 这种方案具有加法同态的性质，也就是说，各参与方可以在不交换任何数据的情况下直接计算对秘密数据求和。什么意思呢？假设还有另一个人B，他也有一个秘密数字 y ,并且和A一起将数据分发给了$S_1, S_2, ..., S_n$,




```
阈值密钥分享（threshold secret sharing）
```

更具体地说，我们可以定义一种名为 (t, n) 阈值密钥分享的方案，此类方案允许任意 t

个参与方将秘密数据解开，但任何不多于 $t-1$ 个参与方的小团体都无法将秘密数据解开。前面提到的那种简单方案其实是 $t=n$ 时的特殊情况。Shamir大神在1979年就提出了阈值密钥分享方案，且该方案支持任意的 t。该方案运作方式如下：假设A想要使用 (t, n) 阈值密钥分享技术将某秘密数字 s 分享给$S_1, S_2, \ldots, S_n$，那么他首先生成一个 $t-1$ 次多项式多项式 $f(x)=a_0+a_1 x+a_2 x^2+\cdots+a_{t-1} x^{t-1}$，其中 a_0 就等于要分享的秘密数字 s，而 $a_1, a_2, \ldots, a_{t-1}$，则是A生成的随机数。随后A只需将 $s_1=f(1), s_2=f(2), \ldots, s_n=f(n)$ 分别发给$S_1, S_2, \ldots, S_n$即可。到了这一步，稍微有点线性代数基础的同学应该很容易看出来，$f(1), f(2), \ldots, f(n)$ 中任意 t 个凑在一起都可以解出，而任意 $t-1$ 个凑在一起都无法得到 a_0 （即 s ）的确切解。通过这一点便达到了 (t, n) 阈值的要求。Shamir密钥分享方法也是满足加法同态的（因为多项式本身满足这一性质），有兴趣的同学可以自己验证一下。

```
乘法
```

现在有A和B分别分享了两个数字 x 和 y，参与方需要算出 x 和 y 的乘积 z 的密钥分享。这时候可以借助前面生成的随机乘积元组。我们先令 $s=x-a$ 以及 $t=y-b$，然后我们可以看到

$$x \times y=(x-a+a) \times(y-b+b)=(s+a) \times(t+b)=s \times t+s \times b+t \times a+c$$

参与方$S_1, S_2, \ldots, S_n$可以联合起来将 s 和 t 的值解开，由于 a 和 b 都是值未知的随机数，因此 s 和 t 的值并不会暴露关于 a 和 b 的信息。上面那个式子中，$s \times t$ 可以直接用公开的 s 和 t 算出来，$s \times b$ 以及 $t \times a$ 的密钥分享则可以用前面的秘密数与公开数的乘法得到，而 c 的密钥分享则是一开始就存在，因此这几项合起来便能得到 $z=x \times y$ 的密钥分享。

- 简短介绍

```
密钥分享的基本思路是将每个数字 x 拆散成多个数 x1,x2,…,xn，
并将这些数分发到多个参与方 S1,S2,…,Sn 那里。
然后每个参与方拿到的都是原始数据的一部分，
一个或少数几个参与方无法还原出原始数据，
只有大家把各自的数据凑在一起时才能还原真实数据。
计算时，各参与方直接用它自己本地的数据进行计算，
并且在适当的时候交换一些数据
（交换的数据本身看起来也是随机的，不包含关于原始数据的信息），
```

```
计算结束后的结果仍以secret sharing的方式分散在各参与方那里，
并在最终需要得到结果的时候将某些数据合起来。
```

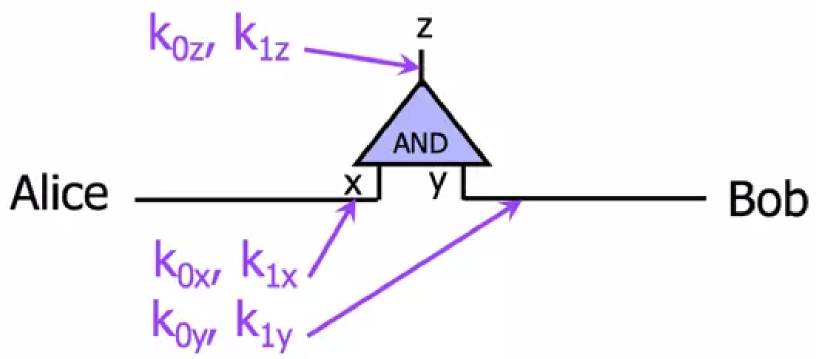
4. garbled circuits

- cryptograghy rationales（密码学基本原理）

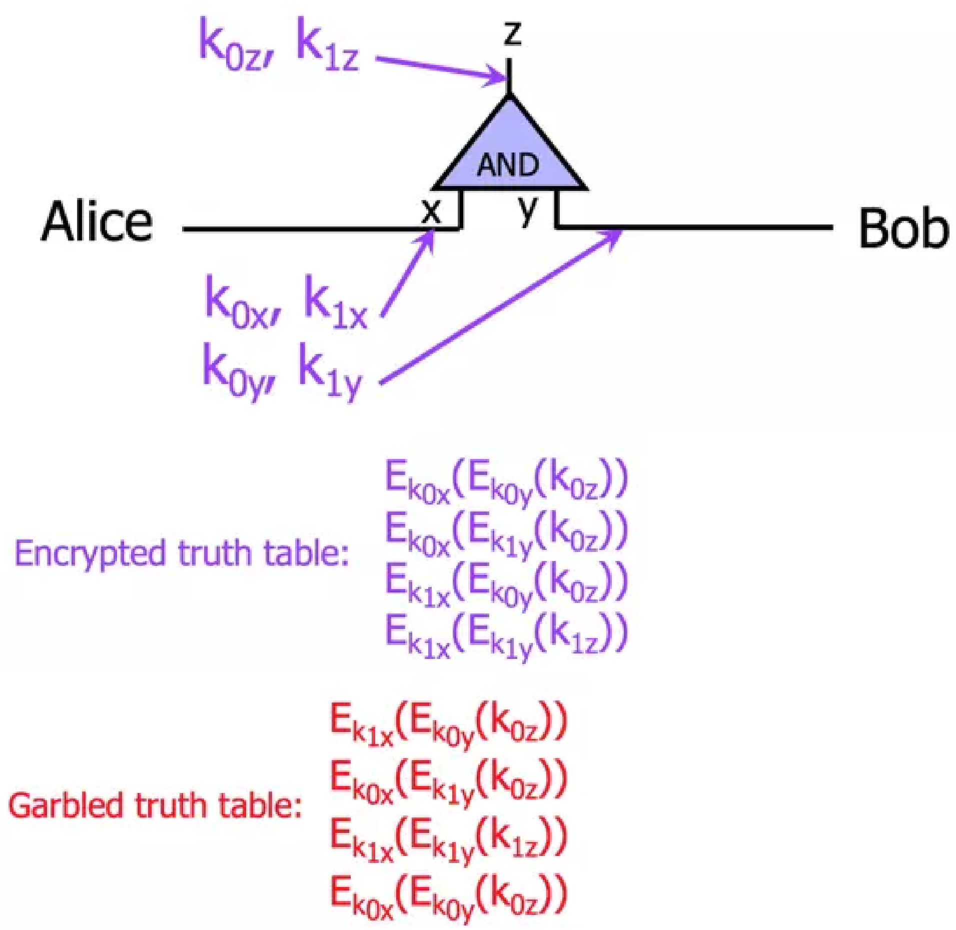
资料来自——混淆电路

```
basic
```

Alice和Bob想计算一个与门。该门两个输入线 x 和 y 和一个输出线 z，每条线有0和1两个可能的值。Alice首先给每条线指定两个随机的key，分别对应0和1。



然后，Alice用这些密钥加密真值表，并将该表打乱后发送给Bob。加密过程就是将真值表中每一行对应的 x 和 y 的密钥加密 z 的密钥。这一加密+打乱的过程，就是混淆电路（garbled circuit）的核心思想。



那Bob收到加密表后，如何计算呢？首先Alice把自己的输入对应的key发给Bob，比如Alice的输入是0，那就发 k_{0x} ，输入是1就发 k_{1x} 。同时把和Bob有关的key都发给Bob，也就是 k_{0y} 和 k_{1y} 。然后Bob根据自己的输入挑选相关的key。由于Bob收到的这些key都是随机数，所以其实并没有任何有效信息泄露。Bob根据收到的 k_x 和自己的 k_y ，对上述加密表的每一行尝试解密，最终只有一行能解密成功，并提取出相应的 k_z 。

Bob将kz发给Alice，Alice通过对比是 k_{0z} 还是 k_{1z} 得知计算结果是0还是1。由于整个过程大家收发的都是密文或随机数，所以没有有效信息泄露。

- 简短介绍

```
先明确一下混淆电路解决的是什么问题。通俗的说，就是一堆人各自拥有其隐私数据，他们想把这些数据合起来算点什么，但又不想把数据交给别人，混淆电路解决的就是此类问题。
```

Part 1b: Basic MP-SPDZ

第 1b 部分：MP-SPDZ 基础

```
2. Your should first install the MP-SPDZ . You are required to implemented a SMPC (secure multiparty computation) version of any sort algorithm. The algorithm will get its input from files named Input-P0-0, Input-P1-0...,
 and reveal the sorted array to all parties. (20%)
```

1. install

- [x] ~~github~~

```
1. Download and unpack the distribution
(https://github.com/data61/MP-SPDZ/releases) mp-spdz-0.3.3.tar.xz.

2. MP-SPDZ documentation:
https://github.com/data61/MP-SPDZ#tldr-binary-distribution-on-linux-or-source-distribution-on-macos
https://mp-spdz.readthedocs.io/en/latest/Compiler.html#

3. MP-SPDZ Publication:
https://ia.cr/2020/521

4. install
Scripts/tldr.sh
./compile.py tutorial
echo 1 2 3 4 > Player-Data/Input-P0-0
echo 1 2 3 4 > Player-Data/Input-P1-0
Scripts/mascot.sh tutorial
```

1. implement a SMPC version of any sort algorithm(实现任意排序算法的安全多方计算版本)

- [ ] implement

```
>>>>>>>>>> 代码：
program.use_edabit(True)
from util import if_else
```

```
from Compiler import types

def maopao(a):
    print_ln("hello: %s", a.reveal())
    n = len(a)
    @for_range(n)
    def _(i):
        @for_range(n)
        def _(j):
            @if_((a[i] > a[j]).reveal())
            def _():
                # print_ln("!!!")
                tmp = a[i]
                a[i] = a[j]
                a[j] = tmp
    return a

n = 2

a = Array(n, sfix)

@for_range_opt(n)
def _(i):
    a[i] = sfix.get_input_from(i)

d = maopao(a)
print_str("%s", d.reveal())

print_ln('Data receive success !!')

>>>>>>>>>> compile
yunxi@yunxi-virtual-machine:~/Desktop/CNNS/classProject/mp-spdz-0.3.3$ ./compile.py merge_and_sort
Default bit length: 64
Default security parameter: 40
Compiling file /home/yunxi/Desktop/CNNS/classProject/mp-spdz-0.3.3/Programs/Source/merge_and_sort.mpc
WARNING: Order of memory instructions not preserved, errors possible
WARNING: Order of memory instructions not preserved, errors possible
Writing to /home/yunxi/Desktop/CNNS/classProject/mp-spdz-0.3.3/Programs/Schedules/merge_and_sort.sch
Writing to /home/yunxi/Desktop/CNNS/classProject/mp-spdz-0.3.3/Programs/Bytecode/merge_and_sort-0.bc
Program requires at most:
         inf integer inputs from player 0
           4 strict edabits of length 41
           4 strict edabits of length 31
         220 bit triples
           4 integer dabits
```

```
         66 virtual machine rounds

>>>>>>>>>> run
yunxi@yunxi-virtual-machine:~/Desktop/CNNS/classProject/mp-spdz-0.3.3$ ./semi-party.x -p 0 -N 2 merge_and_sort
Using security parameter 40
hello: [0, 1]
[1, 0]Data receive success !!
Significant amount of unused dabits of gfp. For more accurate benchmarks, consider reducing the batch size with -b.
Significant amount of unused edaBits of size 31. For more accurate benchmarks, consider reducing the batch size with -b or increasing the bucket size with -B.
Significant amount of unused edaBits of size 41. For more accurate benchmarks, consider reducing the batch size with -b or increasing the bucket size with -B.
Significant amount of unused triples of binary secret. For more accurate benchmarks, consider reducing the batch size with -b.
The following benchmarks are including preprocessing (offline phase).
Time = 0.27738 seconds
Data sent = 11.1609 MB in ~1560 rounds (party 0)
Global data sent = 22.3217 MB (all parties)

yunxi@yunxi-virtual-machine:~/Desktop/CNNS/classProject/mp-spdz-0.3.3$ ./semi-party.x -p 1 -N 2 merge_and_sort
Using security parameter 40
Significant amount of unused dabits of gfp. For more accurate benchmarks, consider reducing the batch size with -b.
Significant amount of unused edaBits of size 31. For more accurate benchmarks, consider reducing the batch size with -b or increasing the bucket size with -B.
Significant amount of unused edaBits of size 41. For more accurate benchmarks, consider reducing the batch size with -b or increasing the bucket size with -B.
Significant amount of unused triples of binary secret. For more accurate benchmarks, consider reducing the batch size with -b.
The following benchmarks are including preprocessing (offline phase).
Time = 0.275165 seconds
Data sent = 11.1609 MB in ~1560 rounds (party 1)
Global data sent = 22.3217 MB (all parties)

>>>>>>>>>> 实现输出到文件
yunxi@yunxi-virtual-machine:~/Desktop/CNNS/classProject/mp-spdz-0.3.3
$./semi-party.x -p 0 -N 2 -IF 'Player-Data/Input' -OF Player-Data/Output merge_and_sort
```

Part 2: Project Task

```
3. Design and implement the solution of a SMPC problem on graphs (e.g.,breadth-first search, depth-first search), and evaluate the security and performance of your algorithm analytically and experimentally. (60%)
```

Part 2a: Project Implementation

- dfs basic theory on paper（论文中基本 dfs 算法）

节点5改名为节点0，并将节点0设为s0

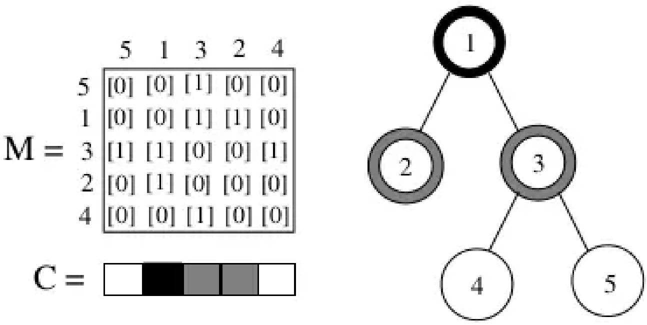


Figure 1: Illustration of the BFS algorithm.

- dfs implementation, design, solution and problem

```python
program.use_edabit(True)
from util import if_else
from Compiler import types

def bfs(M, V, n):
    i2 = Array(1,sint)
    ii = 0
    @for_range(n)
    def _(i):
        i2[0] = 0
        mindis = n
        @for_range(n)
        def _(i1):
            @if_(V[3*i1].reveal()==1)
            def _():
                @if_(V[3*i1+1].reveal()<mindis)
                def _():
                    mindis = V[3*i1+1].reveal()
                    i2[0] = i1
        # print_str(" %s\n",i2[0].reveal())
        # t = sint(0)
        # iii = str(i2.get(t.reveal()))
        # print_ln("%s",iii)
        # ii = int('0')
        @for_range(n)
        def _(j):
            @if_(M[i2[0].reveal()*n+j].reveal()==1)
            def _():
```

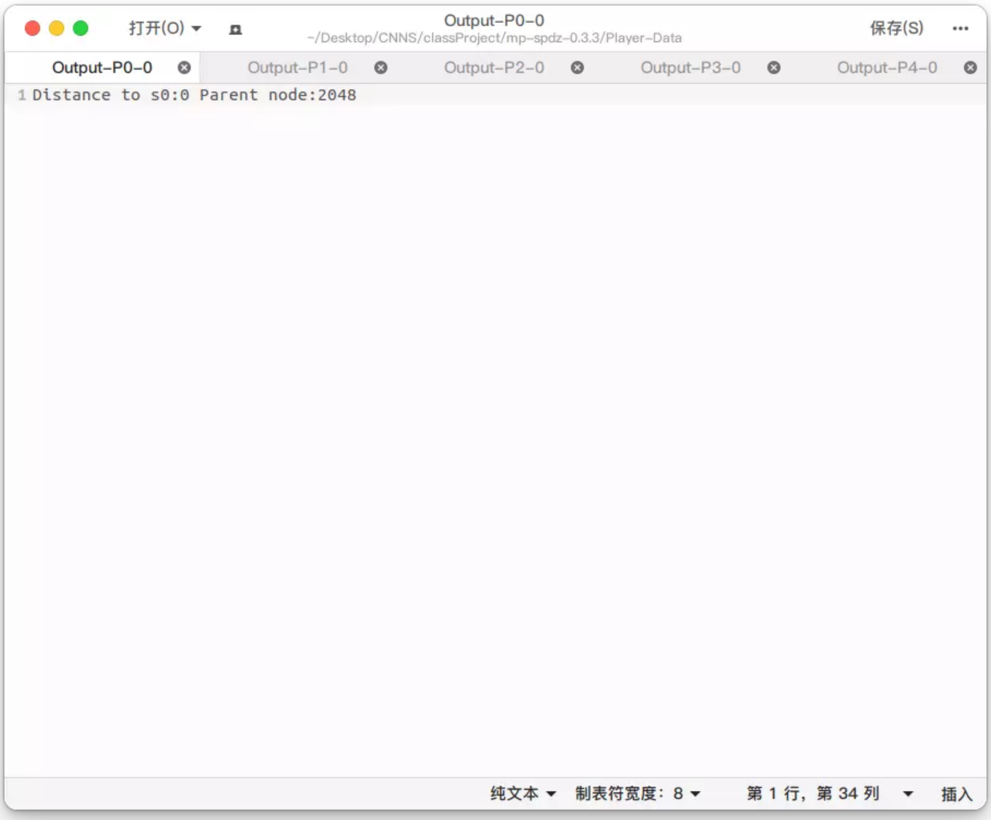
```
                @if_(V[3*j].reveal()==0)
                def _():
                    V[3*j] = 1
                    V[3*j+1] = V[3*i2[0].reveal()+1] + 1
                    V[3*j+2] = i2[0].reveal()
        #print_str("%s %s %s %s %s\n",i1,i2[0].reveal(),V[3*ii].reveal(),V[3
        V[3*i2[0].reveal()] = 2
        #print_str("%s %s %s %s %s\n",i1,i2[0].reveal(),V[3*i2[0]].reveal(),

n = 5
M = Array(n*n, sfix)
V = Array(n*3, sfix)
@for_range_opt(n)
def _(i):
    @for_range_opt(n)
    def _(j):
        M[i*n+j] = sfix.get_input_from(i)

@for_range(n)
def _(i):
        V[i*3] = 0
        V[i*3+1] = 2048
        V[i*3+2] = 2048

V[0] = 1
V[1] = 0

@for_range(n)
def _(i):
        @if_(M[i].reveal()==1)
        def _():
            V[i*3+1] = 1

# print_str("%s\n", M.reveal())
# print_str("%s\n", V.reveal())

bfs(M,V,n)

# print_str("%s", V.reveal())
@for_range(n)
def _(i):
    print_ln_to(i,"Distance to s0:%s Parent node:%s",V[i*3+1].reveal(),V[i*3
```

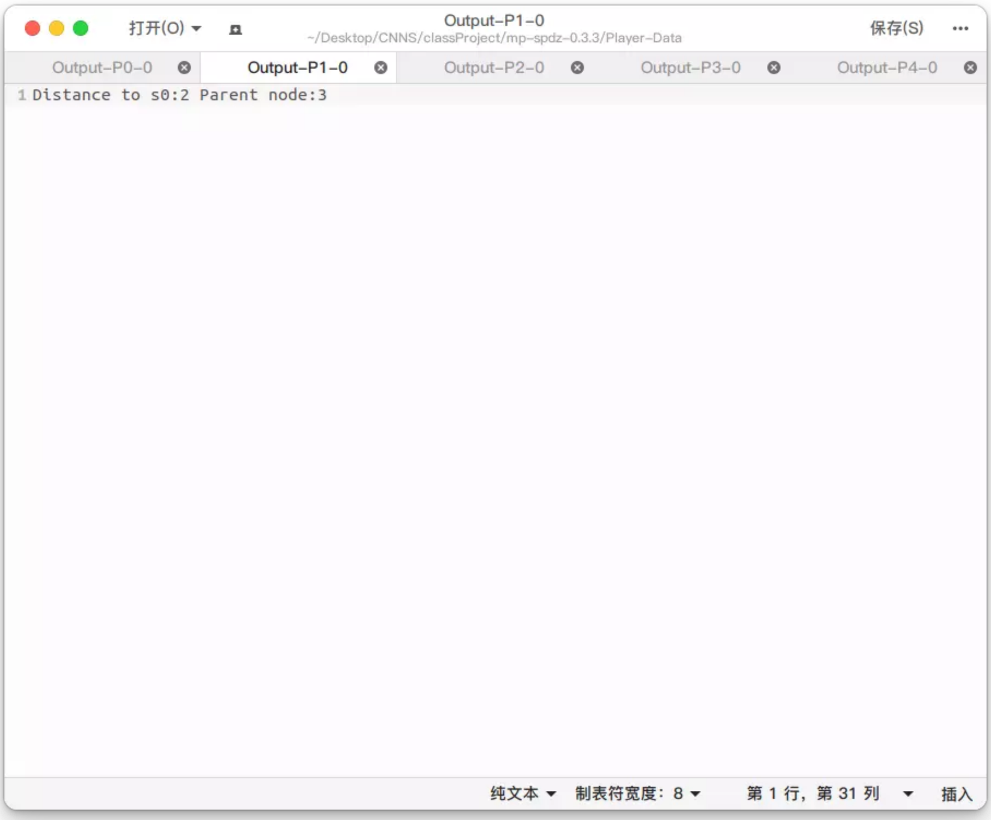
- 命令行 benchmark

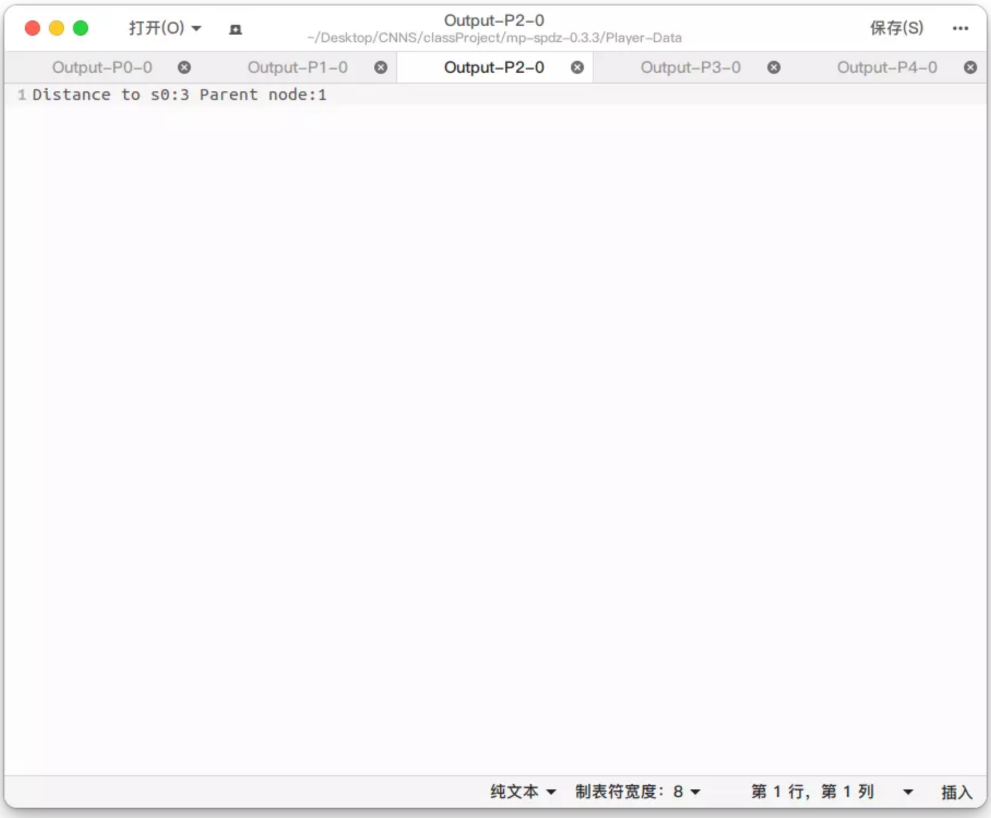
```
./semi-party.x -p 0 -N 5 -IF 'Player-Data/Input' -OF Player-Data/Output bfs
```

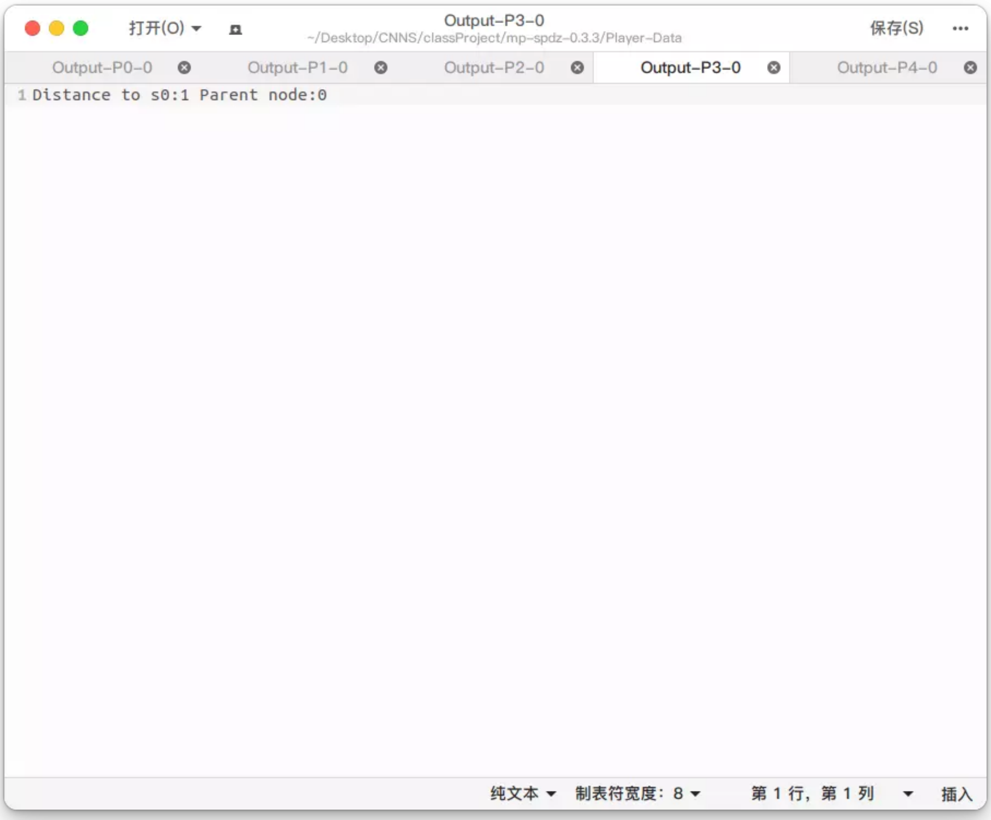
```
yunxi@yunxi-virtual-machine:~/Desktop/CNNS/classProject/mp-spdz-0.3.3$ ./semi-party.x -p 0 -N 5 -IF 'Player-Data/Input' -OF Player-Data/Output bfs
Using security parameter 40
The following benchmarks are including preprocessing (offline phase).
Time = 0.0337145 seconds
Data sent = 0.125796 MB in ~2472 rounds (party 0)
Global data sent = 0.605364 MB (all parties)
```

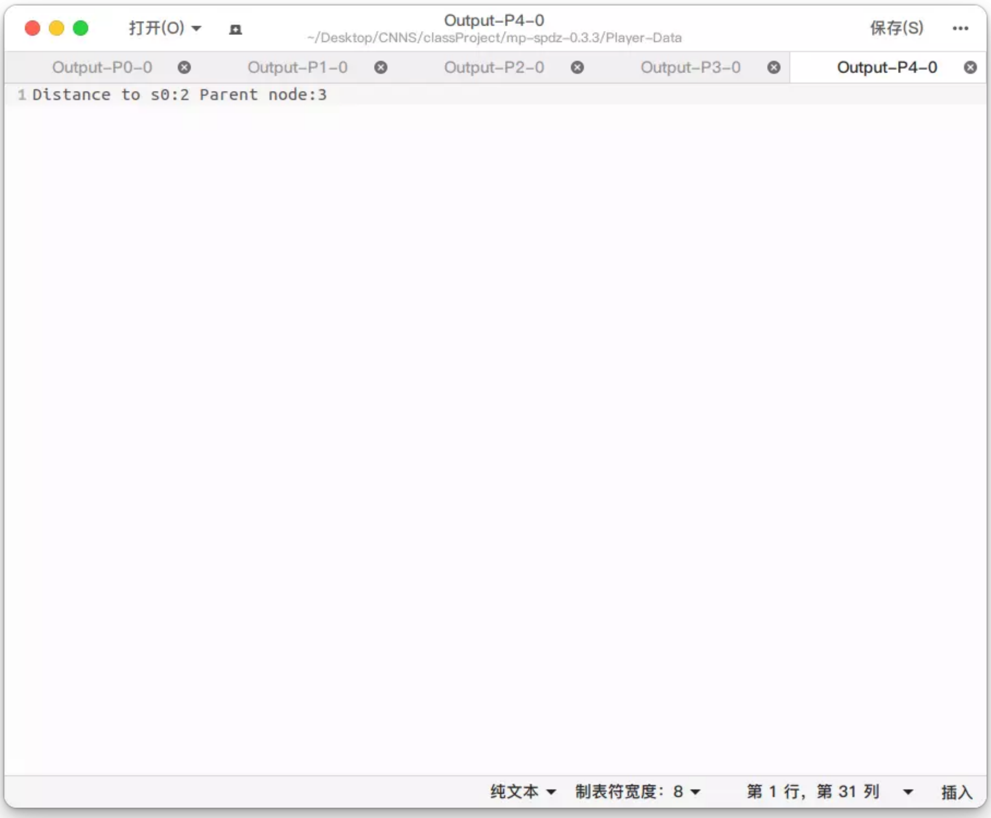
- –OF output file 文件输出




打开(O)
Output–P1–0
~/Desktop/CNNS/classProject/mp–spdz–0.3.3/Player–Data
保存(S)
Output–P0–0
Output–P1–0
Output–P2–0
Output–P3–0
Output–P4–0
1 Distance to s0:2 Parent node:3
纯文本
制表符宽度：8
第 1 行，第 31 列
插入


打开(O)
Output–P2–0
~/Desktop/CNNS/classProject/mp–spdz–0.3.3/Player–Data
保存(S)
Output–P0–0
Output–P1–0
Output–P2–0
Output–P3–0
Output–P4–0
1 Distance to s0:3 Parent node:1
纯文本
制表符宽度：8
第 1 行，第 1 列
插入





Part 2b: evaluate the security and performance of your algorithm analytically and experimentally

- 性能分析

```
算法共有四个步骤：
1. 从 Input-P…… 读入信息生成邻接矩阵，此处分内循环和外循环，循环 n*n 次，如下：
```

```
  @for_range_opt(n)
  def _(i):
    @for_range_opt(n)
      def _(j):
        M[i*n+j] = sfix.get_input_from(i)
2. 初始化数组 C，用于存放每个节点与 s0 的距离，颜色和父亲节点，总共 3*n+2+n 步，如下：
  @for_range(n)
  def _(i):
     V[i*3] = 0
     V[i*3+1] = 2048
     V[i*3+2] = 2048
  V[0] = 1
  V[1] = 0
  @for_range(n)
  def _(i):
     @if_(M[i].reveal()==1)
     def _():
        V[i*3+1] = 1
3. 广度优先搜索，共 n*(n+n) 步，分外循环、找下一个灰色节点、为节点寻找孩子3部分，代码较长，暂不给出。
4. 输出至指定 party，共 n 步，代码如下：
  @for_range(n)
  def _(i):
    print_ln_to(i,"Distance to s0:%s Parent node:%s",V[i*3+1].reveal(),V[i*3+2].reveal())

综上，时间复杂度为 O(n*n + 3*n+2+n + n*(n+n) + n) = O(n*n)
```

- 性能测试

```
将 merge_and_sort 与 bfs 进行比较，可以发现所使用的时间相近。
白色图为merge_and_sort，黑色图为 bfs
```

```
The following benchmarks are including preprocessing (offline phase).
Time = 0.27738 seconds
Data sent = 11.1609 MB in ~1560 rounds (party 0)
Global data sent = 22.3217 MB (all parties)
```

```
yunxi@yunxi-virtual-machine:~/Desktop/CNNS/classProject/mp-spdz-0.3.3$ ./semi-party.x -p 0 -N 5 -IF 'Player-Data/Input' -OF Player-Data/Output bfs
Using security parameter 40
The following benchmarks are including preprocessing (offline phase).
Time = 0.0337145 seconds
Data sent = 0.125796 MB in ~2472 rounds (party 0)
Global data sent = 0.605364 MB (all parties)
```

- 安全性评估

```
1. 程序全部使用存储 sfix 和 sint 加密类型的数组，对数组内数据的读取只能使用 reveal( ) 方法，数据在运算过程中是保密的，难以被拦截。
2. 程序使用 print_ln_to( ) 进行私密输出，每个节点只接收到自己的数据。
```

- 安全性测试

```
如图展示每个节点在文件中的输出。
```