编译技术 2020Project2 报告

杨晨阳 #1700012770
袁野 #1700012821
魏安江 #1700012723
王余越 #1700013031

2020 年 6 月 21 日

目录

1 技术设计

1.1 Motivating example

本次 project 的目标：给定表达式 $O = f(I_1, I_2, ..., I_n)$. （$(O, I_i)$ 是张量或标量，f 表示用其参数构造一个表达式），已知最终 loss 对于 O 的导数 $dO = \frac{\partial loss}{\partial O}$，试求 loss 对于某个输入的导函数，即 $dI_i = \frac{\partial loss}{\partial I_i}$。要求分析得到的求导表达式为多个赋值语句，且左侧下标不可有运算。在接下来的部分，我们使用以下例子解释我们的技术设计：

1. $C\langle M, N\rangle[i][j] = A\langle M, K\rangle[i][k] * B\langle K, N\rangle[k][j] + 1.0$
2. $C\langle N, N\rangle[i][j] = A\langle N, N\rangle[i][j] * A\langle N, N\rangle[i][j]$
3. $C\langle M, N\rangle[i][j] = A\langle M, N\rangle[i+1][j]$
4. $C\langle M, N\rangle[i][j] = A\langle M-R, N-S\rangle[i+r][j+s] * B\langle R, S\rangle[r][s]$

1.2 矩阵求导

首先我们考虑不含下标变换的例子，如例 1、2。考虑 loss=$f(C)$ 本质上是关于 C 所有 item 的一个多元函数，其偏导即对所有 item 分别求偏导，再排列成矩阵。由于我们已知 C 的偏导，对 A 求偏导时，可以利用 dC，根据链法则求导。

例如，我们想要求例 1 中 $dA[i_0][j_0]$ 的值。在所有 $C[i][j] = \sum_k A[i][k] * B[k][j] + 1.0$ 的式子中，只有 $C[i_0][j], \forall j$ 含有 $A[i_0][j_0]$ 项，它们的导数为 $B[j_0][j], \forall j$。根据链法则，需要乘上矩阵 C 对应 item 的导数并求和，即

$$dA[i_0][j_0] = \sum_j dC[i_0][j]B[j_0][j]$$

利用爱因斯坦求和法的标记，我们得到

$$dA\langle M, K\rangle[i][k] = dC\langle M, N\rangle[i][j] * B\langle K, N\rangle[k][j]$$

对例 2，我们同样计算可得

$$dA\langle N, N\rangle[i][j] = dC\langle N, N\rangle[i][j] * A\langle N, N\rangle[i][j] + A\langle N, N\rangle[i][j] * dC\langle N, N\rangle[i][j]$$

事实上，由于每个等式只是标量之间的关系，我们可以直接用对标量的求导方法进行求导，再根据链法则乘上对应的输出导数，进行求和。我们只需要在底层实现对 +、-、*、/以及单个标量的求导，并在下一步对含有下标变换的例子额外处理即可。

例如，例 2 实质上等价于 $c = a * a$，对 a 求导结果即为 $c = 1 * a + a * 1$。由于链法则，需要在每次运算到 $\frac{\delta a}{a} = 1$ 时，乘上合适的 dC 项。在这一例子中，$\frac{\delta A[i][j]}{A[i][j]}$ 即需要乘上 $dC[i][j]$，如此便可得到上式计算结果。

当涉及到下标变换时，我们的处理思路依旧一致，只不过需要注意分析求导时涉及的项。在例 3 中，我们试图对 $A[i_0][j_0]$ 求导，则需要考虑 $C[i_0-1][j_0]=A[i_0][j_0]$ 一式，而此时链法则需要乘上的项则为 $C[i_0-1][j_0]$。而在例 4 中，对 $A[i_0][j_0]$ 求导，我们则需要考虑 $C[i_0-r][j_0-s]=A[i_0][j_0]*B[r][s]$ 等方程，此时链法则需要乘上的项为 $\{C[i_0-r][j_0-s]\}$。

可以看到，对所有线性的下标变化，我们都可以归结为解线性方程组，而这个问题可以通过求矩阵 Smith 标准型解决，具体算法详见下一部分。

而对于非线性的下标变化，如

$$B\langle 32\rangle[i]=A\langle 2,16\rangle[i//16,i\%16]$$

我们需要解如下方程:

$$i_0=i//16$$
$$j_0=i\%16$$

亦即

$$i=16*i_0+s,0\leq s<16$$
$$i=16*r+j_0$$

容易反解得到，$i=16*i_0+j_0$。

1.3 下标变换

1.3.1 求矩阵史密斯标准型

在下标变换中，需要求线性方程组的整数解。
考虑如下线性方程组:

$$A_{11}x_1+A_{12}x_2+...+A_{1m}x_m=b_1$$
$$A_{21}x_1+A_{22}x_2+...+A_{2m}x_m=b_2$$
$$......$$
$$A_{n1}x_1+A_{n2}x_2+...+A_{nm}x_m=b_n$$

A 为元素均为整数的 $N\times M$ 矩阵，x_i 为整数变量，目标 b_i 为整数。
原问题可用矩阵简化表达为

$$A\boldsymbol{x}=\boldsymbol{b}$$

要求出这组线性方程的解集，需要将矩阵 A 转化成史密斯标准型。史密斯标准型是一种对角矩阵，形式为

$$S=diag(\alpha_1,\alpha_2,...,\alpha_R,0,...,0)$$

同时史密斯标准型需要满足 $\alpha_i|\alpha_{i+1}, 1 \le i < r$ a|b 指 a 整除 b
这个对角矩阵非零 α 的个数等于原矩阵 A 的秩。对于每一个非零整数矩阵 $A \in \mathbb{Z}^{N\times M}$，都存在可逆矩阵 $U \in \mathbb{Z}^{N\times N}$ 和 $V \in \mathbb{Z}^{M\times M}$，使得

$$S = UAV$$

这个 $S \in \mathbb{Z}^{N\times M}$ 就是 A 的史密斯标准型。使用史密斯标准型，原问题可化为

$$S\boldsymbol{x'} = \boldsymbol{b'}$$

$$\boldsymbol{x} = V\boldsymbol{x'}, \boldsymbol{b'} = U\boldsymbol{b}$$

定义矩阵 $C \in \mathbb{Z}^{N-R}$, 其中 $C_{i,j} = U_{R+i,j}$
定义矩阵 $I = VS^{\dagger}U$, 其中 $S^{\dagger} = diag(1/\alpha_1, 1/\alpha_2, ..., 1/\alpha_R, 0..., 0)$
经证明，若 $C\boldsymbol{b} = 0$ 且 $I\boldsymbol{b} \in \mathbb{Z}^N$ 则原方程组有解。特别地，如果 A 满秩，则有唯一整数解 $\boldsymbol{x} = I\boldsymbol{b}$。如果 A 不满秩，则有无数整数解 $\boldsymbol{x}$，满足 $\boldsymbol{x} = I\boldsymbol{b} + K\boldsymbol{z}$, 其中 $z \in \mathbb{Z}^{M-R}$。

1.4 编译知识

本次 project 用到的主要知识有：文法分析中的消除左递归和左公因子（因为我们使用的 antlr 是 LL parser），遍历抽象语法树，添加语义动作构建 SDT（在 json 转 IR，IRMutator 生成求导两步均有用到），中间代码生成。在 IR 这个抽象层面进行代码变换，实现求导，这个过程再次为我们展示了编译技术的魅力。

2 实现流程

2.1 Json2IR

这部分任务与 project1 相似，只不过不需要展开循环表达式，而是保留原始的运算符号，提供给 mutator 做求导。例如 $A = B * C$ 构建出的语法树就为 MOVE(A, MUL(B, C))。核心代码如下：

```
    antlrcpp::Any visitRhsE(projectExprParser::RhsEContext *ctx) override {
        Expr expr = ctx->children[0]->accept(this);
        std::pair<std::string, Expr> op_pair = ctx->children[1]->accept(this);
        if (op_pair.first == "")
            return expr;
        if (op_pair.first == "+")
            return Binary::make(data_type, BinaryOpType::Add, expr, op_pair.second);
        else if (op_pair.first == "-")
            return Binary::make(data_type, BinaryOpType::Sub, expr, op_pair.second);
        assert (false);
    }
```

```
    antlrcpp::Any visitRhsER(projectExprParser::RhsERContext *ctx) override {
        size_t n = ctx->children.size();
        if (n == 0)
            return std::pair<std::string, Expr>("", NULL);
        std::string op = ctx->children[0]->getText();
        Expr expr = ctx->children[1]->accept(this);
        return std::pair<std::string, Expr>(op, expr);
    }
    antlrcpp::Any visitRhsI(projectExprParser::RhsIContext *ctx) override {
        size_t n = ctx->children.size();
        Expr expr;
        std::string text;
        std::pair<std::string, Expr> op_pair;
        if (n == 2) {
            expr = ctx->children[0]->accept(this); // should return a var
            op_pair = ctx->children[1]->accept(this);
        } else {
            expr = ctx->children[1]->accept(this);
            expr = Cast::make(expr->type(), expr->type(), expr);
            op_pair = ctx->children[3]->accept(this);
        }
        if (op_pair.first == "")
            return expr;
        if (op_pair.first == "*")
            return Binary::make(data_type, BinaryOpType::Mul, expr, op_pair.second);
        else if (op_pair.first == "//" || op_pair.first == "/")
            return Binary::make(data_type, BinaryOpType::Div, expr, op_pair.second);
        else if (op_pair.first == "%")
            return Binary::make(data_type, BinaryOpType::Mod, expr, op_pair.second);
        else throw (std::invalid_argument("Unknown Error"));
    }
    antlrcpp::Any visitRhsIR(projectExprParser::RhsIRContext *ctx) override {
        size_t n = ctx->children.size();
        if (n == 0)
            return std::pair<std::string, Expr>("", NULL);
        std::string op = ctx->children[0]->getText();
        Expr expr = ctx->children[1]->accept(this);
        return std::pair<std::string, Expr>(op, expr);
    }
    antlrcpp::Any visitTRef(projectExprParser::TRefContext *ctx) override {
        visitChildren(ctx);
        std::string name = ctx->ID()->getText();
        Expr var = Var::make(data_type, name, idx, shape);
        idx.clear();
        shape.clear();
        if (vars.find(name) == vars.end())
            vars.insert(std::pair<std::string, Expr>(name, var));
        return var;
```

```
    }
    antlrcpp::Any visitSRef(projectExprParser::SRefContext *ctx) override {
        visitChildren(ctx);
        std::string name = ctx->ID()->getText();
        Expr var = Var::make(data_type, name, {}, {});
        if (vars.find(name) == vars.end())
            vars.insert(std::pair<std::string, Expr>(name, var));
        shape.clear();
        return var;
    }
```

2.2 IR 求导

对于最底层的变量，当其为所求导对象时，求导后值为 1，且需要乘上对应的输出导数；否则求导后值为 0。我们使用 is_left 判断变量是否为右值。

```
Expr IRMutator::visit(Ref<const Var> op) {
    if (is_left) {
        left = Var::make(op->type(), "d" + op->name, op->args, op->shape);
        set_left = true;
    } else if (set_left) {
        if (op->name == grad_to) {
            if (!grad_set) {
                // should rename args later
                grad = Var::make(op->type(), "d" + op->name, op->args, op->shape);
                grad_set = true;
            }
            // should be modified later for index transformation
            // before: args, transformed: new_args
            std::vector<Expr> new_args;
            for (auto arg : left.as<Var>()->args) {
                new_args.push_back(mutate(arg));
            }
            return Var::make(left.as<Var>()->type(), left.as<Var>()->name,
                new_args, left.as<Var>()->shape);
        } else {
            return FloatImm::make(op->type(), 0.0);
        }
    }
    return Var::make(op->type(), op->name, op->args, op->shape);
}
```

对二元运算，我们使用基础的求导知识，并优化含 0 的乘积项。以乘法为例：

```
    Expr res;
    Expr new_a = mutate(op->a);
```

```
    Expr new_b = mutate(op->b);
    bool zero_flag_l = (new_a.node_type() == IRNodeType::FloatImm);
    bool zero_flag_r = (new_b.node_type() == IRNodeType::FloatImm);
    ......
    case BinaryOpType::Mul: {
        Expr item1 = Binary::make(op->type(), op->op_type, new_a, op->b);
        Expr item2 = Binary::make(op->type(), op->op_type, op->a, new_b);
        if (zero_flag_l && zero_flag_r) {
            return FloatImm::make(op->type(), 0.0);
        } else if (zero_flag_l) {
            res = item2;
        } else if (zero_flag_r) {
            res = item1;
        } else {
            res = Binary::make(op->type(), BinaryOpType::Add, item1, item2);
        }
        break;
    }
```

和下标变换的衔接将于下一部分具体说明。

2.3 下标变换

2.3.1 求矩阵史密斯标准型

定义如下 matrix 类，包含矩阵乘、元素查找、行列变换等操作

```
    class matrix {
public:
    int n_rows; //number of rows of the matrix
    int n_cols; //number of columns of the matrix
    int* x;     //array to store the content of the matrix
    matrix(int r,int c);
    void insert(int _x[]);
    //use array to assign element values to the matrix
    void operator=(matrix t);
    //use another matrix to assign element values to the matrix
    int& value(int i,int j);
    //return the (row i, column j) element of matrix
    bool empty_after(int a,int& row,int& col);
    //check if there exists non-zero element(i,j) where i>=a or j>=a,
    //if so, return its row and column as pivot element
    bool undiv_in_row(int a,int& col);
    //check if there exists un-divisible element(a,j) where j>a,
    //if so, return its column
    bool undiv_in_col(int a,int& row);
    //check if there exists un-divisible element(i,a) where i>a,
```

```
    //if so, return its row
    bool empty_in_row(int a,int& col);
    //check if there exists non-zero element(a,j) where j>a,
    //if so, return its column
    bool empty_in_col(int a,int& row);
    //check if there exists non-zero element(i,a) where i>a,
    //if so, return its row
    void swap(int direction,int a,int b);
    //if direction = 1, swap the a_th and b_th columns
    //if direction = 0, swap the a_th and b_th rows
    void alter_row(int row1,int row2,int sigma,int tau,int gama,int alpha);
    //simutaneously assign row1 := sigma * row1 + tau * row2,
    //                     row2 := -gama * row1 + alpha * row2
    void alter_col(int col1,int col2,int sigma,int tau,int gama,int alpha);
    //simutaneously assign column1 := sigma * column1 + tau * column2,
    //                     column2 := -gama * column1 + alpha * column2
    matrix operator* (matrix x);
};
```

对原矩阵 A 的不同大小右下角子矩阵，寻找非零元作为主元，并逐个消去主元所在行列非对角位置的非零元。当迭代到算法终止时，得到的矩阵 S 满足对角矩阵性质。

```
int diagonalize(int a,matrix& S,matrix& U,matrix& V) {
    int row,col;
    while(!S.empty_after(a,row,col)) {
        S.swap(1,a,col);
        V.swap(1,a,col);
        S.swap(0,a,row);
        U.swap(0,a,row);

        while(true) {
            int changed = false;
            while(!S.undiv_in_col(a,row)) {
                changed = true;
                int beta,sigma,tau;
                extended_euclidean(S.value(a,a),S.value(row,a),sigma,tau,beta);
                int gama = S.value(row,a)/beta;
                int alpha = S.value(a,a)/beta;
                S.alter_row(a,row,sigma,tau,gama,alpha);
                U.alter_row(a,row,sigma,tau,gama,alpha);

            }
            while(!S.empty_in_col(a,row)) {
                changed = true;
                int f = S.value(row,a)/S.value(a,a);
                S.alter_row(a,row,1,0,f,1);
                U.alter_row(a,row,1,0,f,1);

```

```
            }
            while(!S.undiv_in_row(a,col)) {
                changed = true;
                int beta,sigma,tau;
                extended_euclidean(S.value(a,a),S.value(a,col),sigma,tau,beta);
                int gama = S.value(a,col)/beta;
                int alpha = S.value(a,a)/beta;
                S.alter_col(a,col,sigma,tau,gama,alpha);
                V.alter_col(a,col,sigma,tau,gama,alpha);

            }
            while(!S.empty_in_row(a,col)) {
                changed = true;
                int f = S.value(a,col)/S.value(a,a);
                S.alter_col(a,col,1,0,f,1);
                V.alter_col(a,col,1,0,f,1);

            }
            if(!changed)
                break;
        }
        a+=1;
    }
    return a;
}
```

要由普通的对角矩阵得到满足史密斯标准型对角元素关系的矩阵，还需要对对角元素求公因子，消元等。不断扫描对角元素并调用对角化函数对矩阵调整，当矩阵对角非零元素全部满足性质 $a_{i,i}|a_{i+1,i+1}$ 时，返回所得的标准型及 U,V 矩阵

```
void transform(matrix& S, matrix& U,matrix& V) {
    int a = 0;
    a = diagonalize(a,S,U,V);

    while(1) {
        int R = a-1;
        bool complete = true;
        for(int a=0;a<=R;a++) {
            if(S.value(a,a)<0) {
                for(int i=0;i<S.n_rows;i++)
                    S.value(i,a) = -S.value(i,a);
                for(int i=0;i<V.n_rows;i++)
                    V.value(i,a) = -V.value(i,a);
            }
            if(a<R && S.value(a+1,a+1)%S.value(a,a)!=0) {
                for(int i=0;i<S.n_rows;i++)
                    S.value(i,a) = S.value(i,a) + S.value(i,a+1);
```

```
                for(int i=0;i<V.n_rows;i++)
                    V.value(i,a) = V.value(i,a) + V.value(i,a+1);
                a = diagonalize(a,S,U,V);
                complete = false;
                break;
            }
        }
        if(complete==true)
            break;
    }
}
```

2.3.2 使用史密斯标准型求解的例子

首先，我们给出原矩阵的例子。它对应于 case6。当我们识别出待求导的下标是中含有二元表达式的时候，比如这里的 p+r 和 q+s，那么我们会为它们分别重命名为 z0 和 z1，由于 B 是求导对象，它被放到等式左侧 dB[n][c][z0][z1] 之后，下标就不再带有二元式了。此时，我们为它建立一个矩阵，记录这样的对应关系，譬如矩阵的第三行、第四行分别代表着 $z0 = p + r$, z1=q+s。

A[n][k][p][q] = B[n][c][p + r][q + s] * C[k][c][r][s];

- 首先构造transform matrix:它是从n,c,p,r,q,s变换到n,c,p+r,q+s的矩阵。其中，z0,z1是重命名变量。我们希望通过这个矩阵，能反解n,c,p,r,q,s的结果。注意到两边维度并不对应，所以我们需要之后在等式左边的Y中引入新的变量。

$$\begin{bmatrix} n \\ c \\ z0 \\ z1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} n \\ c \\ p \\ r \\ q \\ s \end{bmatrix}$$

接下来，我们调用 2.3.1 节的求解史密斯标准型的算法

Smith标准型算法的计算结果

$$A=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad U=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$V=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad U*A*V=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

之后，我们需要从数学上的计算结果，得到 Index 之间的对应关系，从而进行下标变换。但是由于维度上的不对应，我们需要引入 2 个新的变量 r0，r1。

变量对应

- 首先将左边的Y进行扩展，引入新变量r0, r1。接着使用扩展后Y与U，V矩阵得到变量之间的对应关系。

$$X = \begin{bmatrix} n \\ c \\ p \\ r \\ q \\ s \end{bmatrix} \Leftrightarrow VY' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n \\ c \\ z0 \\ z1 \\ r0 \\ r1 \end{bmatrix}$$

由此得到： p = z0 – r0, r=r0, q = z1 – r1, s = r1
dB[n, c, z0, z1] = dA[n, k, z0-r0, z1-r1] * C[k, c, r0, r1]

我们将上述对应的关系存放在 map 中，之后遍历到对应的 index 节点后，使用相应的 mutator 将对应下标进行替换，比如将原来的 p 和 q 分别替换成了 z0-r0 和 z1-r1, 将原来的 r 和 s 替换成了新的 r0 和 r1。当前版本的代码仍有一些需要改进的地方，比如现在只支持变换后的矩阵中一行最多 1 个 1 和一个-1 或者只有一个 1 其他全是 0 的情况。接着我们将对应的变量进行减法（z0-r0,z1-r1），生成一个新的 index 变量，或是一一对应的（n,c,r,s），放入 map 中与待替换的变量对应。从 case6 来看，整个过程是自动化的，但是需要说明的是，由于时间所限，目前由待求导表达式到变换矩阵的建立过程还不够通用，比如无法处理下标中有整数相乘的例子比如 B[2*n, c*3]，由于测试样例的缺少，也无法保证每种线性变换都能把矩阵建立正确，或许以后有机会可以重新设计相关数据结构，将矩阵的建立、与原始变量的替代这一复杂的过程适用于更多、更广的样例。

2.3.3 非线性下标变换的处理

由于史密斯标准型仅能解决线性变换的例子，所以针对 case8 中含有整除运算和取模运算的情况，我们的解决方案是在 IR 节点处直接进行变换操作，不再交给通用求解器求解。我们目前只支持整除符号和取模运算的除数相等的情况，具体的下标变换方式在 1.2 节已经进行了详细描述。

2.4 整合

代码的整合主要是将不同模块的接口对接起来，在通过 IRMutator 对原式进行求导后，将生成的求导后的式子打印出来，并传递给 project1 的部分展开循环，得到最终的表达式。

3 实验结果

10 个 case 全部通过。对应的 commit SHA 为 ef35be945c65c191a6d73ae8f7b0e0a7b4bc6e78

```
[ 97%] Building CXX object project1/CMakeFiles/test1.dir/kernels/kernel_case6.cc.o
[ 97%] Building CXX object project1/CMakeFiles/test1.dir/kernels/kernel_case7.cc.o
[ 98%] Building CXX object project1/CMakeFiles/test1.dir/kernels/kernel_case8.cc.o
[ 98%] Building CXX object project1/CMakeFiles/test1.dir/kernels/kernel_case9.cc.o
[100%] Building CXX object project1/CMakeFiles/test1.dir/kernels/kernel_example.cc.o
[100%] Linking CXX executable test1
[100%] Built target test1
david@ubuntu:~/Desktop/CompilerProject/build$ cd project2
david@ubuntu:~/Desktop/CompilerProject/build/project2$ ./test2
Random distribution ready
Case 1 Success!
Case 2 Success!
Case 3 Success!
Case 4 Success!
Case 5 Success!
Case 6 Success!
Case 7 Success!
Case 8 Success!
Case 9 Success!
Case 10 Success!
Totally pass 10 out of 10 cases.
Score is 15.
david@ubuntu:~/Desktop/CompilerProject/build/project2$ git rev-parse HEAD
ef35be945c65c191a6d73ae8f7b0e0a7b4bc6e78
david@ubuntu:~/Desktop/CompilerProject/build/project2$
```

4 小组分工

- 杨晨阳：基本的 IRMutator/求导器，以及 IRPrinter 的改造。
- 袁野：json 到 IR 的转换，solution 代码整合。
- 魏安江：整体思路构建、实现下标变换。
- 王余越：输入任意矩阵，求史密斯标准型 ($S = U \times A \times V$)

5 Bug 提交

我们小组发现了一处 bug，并提交了相应的 Pull Request: https://github.com/pku-compiler-design-spring/CompilerProject-2020Spring/pull/4