怎样实现一个完整的DAG图优化

1. **问题提出**

教材上讲到DAG图算法十分简略，在很多问题上模棱两可，在算法实现中会出现很多问题。如下提出实际实现中遇到的问题：

1. 基本块的划分问题

教材中所说的基本块划分方案以跳转为分隔条件，同时DAG图优化后导出的中间代码顺序相对于优化前的中间代码顺序有很大的改变，可以说是完全打乱了代码顺序。这使得之前对顺序有严格要求的代码实现出现问题。例如连续的输入或输出语句，经过优化后无法保证顺序的正确。

1. 导出中间代码时哪些变量需要保留

在优化前的代码中，会出现很多可以通过优化删去的中间变量，同时又有一些中间变量会在其他部分使用，需要保留。这一部分需要我们进一步思考。

1. 关于修改数组内的值的问题

在教材中介绍的方法中，形如a[x] = b的中间代码重写为a = x []= b，但该操作会出现问题。举例表示：

假设有中间代码两项，根据规则我们形成DAG图中，变量各占一个结点，而变量所在的结点从结点的父结点转移到了结点的父结点。导出中间代码时只会保留一句，而丢失了之前的赋值。

1. 当结点不存在变量时如何导出中间代码

我们使用如下例子说明该问题：

有中间代码此时我们创建结点1表示常数1，创建父结点2表示变量a。在这之后的第二句，我们创建结点3作为结点2的父结点，并将a从结点2转移至结点3。当我们导出中间代码时发现，结点2中的变量为空，导致程序无法知道将1+1的值赋给谁。

1. **解决方案**

针对之前提出的四个问题提出可行的解决方案。

1. 基本块的划分问题

我们知道在基本块内我们无法保证代码的正确顺序，但我们能够保证基本块与基本块之间的相对顺序的正确。因此我们遇到在需要保证中间代码正确顺序的代码块时将其划分为不同的基本块。

1. 导出中间变量时哪些变量需要保留

首先程序中声明的变量是需要保留的。其次，如果某一变量跨越了基本块，那么它可能是需要保留的，例如，a+b的值放在临时变量t0中，由于1）中解决方案，计算a+b和输出将会被划分如不同的基本块，此时如果我们不对t0做更新操作，就不能保证输出的正确性。不过也存在变量跨越了基本块却不需要保留的情况。保守的方案是：用户定义的变量和跨越基本块的临时变量都需要保留。

1. 关于修改数组内的值的问题

为了防止第一个数组赋值被删去，一个保守的方案是不采用教材中对数组赋值操作的改写，将其划分为不同的基本块。

1. 当结点不存在变量时如何导出中间代码

为了解决该问题，之前尝试过保留变量的移动路径，但由于一直存在问题，没有成功实现。当前采用的方案是创建一个新的临时变量填入这个结点。如果为空的结点是叶子结点，还需要输出中间语句来保证新的临时变量获得原叶子结点的值。

1. **数据结构与算法描述**
2. 结点数据结构和需要维护的结点表

typedef struct dag\_node {

bool inQ = false; // 结点是否在队列中，启发式导出时使用

bool leaf; // 结点是否为叶子结点

string leaf\_var\_name; // 叶子结点的变量名，非叶子结点时为空

optype op; // 该结点的操作符，叶子结点和直接赋值为\_null

dag\_node\* left; // 左子结点

dag\_node\* right; // 右子结点

vector<dag\_node\*> parent; // 父结点数组

} \*dag\_link;

vector<dag\_link> node\_list; // 结点数组，记录所有创建的结点

map<string, dag\_link> node\_map; // 结点表，记录每个变量对应的结点

map<dag\_link, vector<string>> reverse\_node\_map; // 逆结点表，由结点对应变量

1. 算法描述
2. 将源中间代码分块，分块规则为：当该行代码非加、减、乘、除、赋值、取数组时，将当前行作为入口语句，将下一行作为入口语句。目的是将当前行划分为单独的一块。当前行包含跳转标签时，将当前行作为入口语句。划分块的同时记录代码中包含的变量。
3. 根据分块结果和从代码中抽取的变量得到变量与基本块序号的对应关系。如果某个变量是用户定义的变量，或是该变量出现在多个基本块中我们将其加入到保留变量池中。
4. 对于划分的基本块中每个仅有一行的基本块，直接打印，其余超过一行的基本块，对其构建DAG图优化。循环基本块内的代码，对于每一行代码：
   1. 寻找第一个操作数v1对应的结点。如果没有找到则创建结点p，p结点leaf标记为真，leaf\_var\_name填入该操作数的名字。将p加入node\_list，将v1与该p的对应关系填入node\_map。
   2. 如果是直接赋值语句，检查v1对应的结点p。如果是非叶子结点，则将左值v0和p的对应关系填入node\_map。此处不关心v0是否在之前就已经有对应结点。如果是叶子结点p，则遍历检查p的父结点中操作符为\_null的结点，如果存在则将v0与该结点的对应关系填入node\_map。否则创建新的结点q，q的leaf标记为假，操作符为\_null，并将p作为q的左子结点。将q加入node\_list，将v0与q的对应关系加入node\_map。
   3. 如果是非直接赋值语句，使用a)中描述的方法查找或创建结点。寻找两操作数共有的操作符为当前操作符的父结点，如果没有则创建父结点。将左值和该结点的对应关系填入node\_map。

在DAG图构造完成后，能够保证：叶子结点至多有一个变量；数组，常数和常量只出现在叶子结点；表示直接赋值的结点结构中必定有一个叶子结点。

1. 将node\_map的对应关系翻转，得到reverse\_node\_map。遍历node\_list中的结点，如果reverse\_node\_map中未出现该结点，说明该结点没有对应任何变量。此时创建新的临时变量t0填入该结点。进一步检查这一结点是否为叶子结点，如果为叶子结点还应当增加赋值语句t0=leaf\_var\_name，并填入保留变量池，保证t0的值是正确的且需要保留。

这一步完成后保证node\_list中的每一个结点在reverse\_node\_map中都有一组非空的变量与其对应。

1. 启发式算法得到结点导出队列。
   1. 将结点p置为空。遍历node\_list，找到一个leaf为假，inQ为假，所有父结点的inQ为真或没有父结点的变量，将其inQ置为真，并放入队列。
   2. 如果结点p为空，导出完成。
   3. 将p置为p的左结点。如果p为叶子结点或p存在不在队列中的父结点，返回a)。否则将p的inQ置为真，放入队列，返回c)。
2. 反向循环结点队列。对于该结点中的变量，如果不存在保留变量池中的变量，则跳过该结点，对下一个结点进行导出。否则，使用calculate函数得到该结点的左结点变量名。如果存在右结点，同样使用calculate函数得到该结点的右结点变量名。根据结点的op，选取结点中的任一保留变量作为左值打印中间代码，并将该结点的其余结点的值同时更新。

此处解释calculate函数的实现：

* 1. 如果该结点为叶子结点，返回该结点对应的变量名。
  2. 如果该结点操作符为\_null，即为直接赋值语句，返回该结点的子结点对应的变量名。
  3. 如果该结点中存在保留变量，则返回该保留变量。
  4. 否则递归调用calculate得到左结点变量名和右结点变量名。任取该结点中一个变量放入保留变量中，并赋值为获得的左右结点变量名根据结点操作符的运算结果。返回该变量。

1. 最后根据保留变量更新符号表。删去没有在保留变量中出现的变量，保留变量中新出现的临时变量作为全局变量存入符号表。
2. **与活跃变量分析的结合**

这一DAG图优化算法可以和活跃变量分析相结合进一步缩小保留变量的范围。首先进行活跃变量分析，得到每一基本块的out集，那么每一基本块的保留变量就变为用户定义变量和out集中的临时变量。在更新符号表时，可以将所有块的保留变量取并集作为完整的保留变量池，用这一保留变量池更新符号表。由于时间原因该想法未能实现。