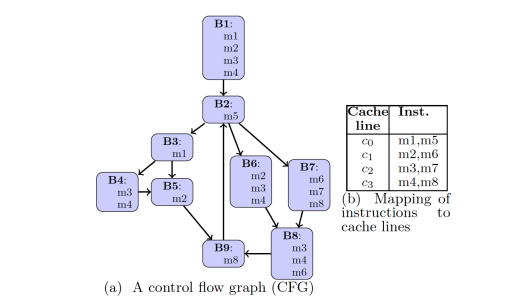
Introduction to Cache Analysis. The purpose of this assignment is to develop algorithms to compute, as precisely yet as quickly as possible, a safe overestimate of the cache misses that can happen in a given computer program. For simplicity, we can assume that we are dealing with a simplistic program that can be converted into a control flow graph (CFG) as shown in the figure below.

缓存分析简介。 这个作业的目的是实现一种既快又准地计算在给定的计算机程序中发生缓存未命中的估计。

为简单起见，我们可以假定我们正在处理一个简单的程序，可以将其转换为控制流程图（CFG），如下图所示。



The CFG shows the program’s control points as its nodes. The nodes in figure (a) above show which memory blocks are needed at each control point. For instance, the initial control point denoted as node B1 needs to load memory blocks m1, m2, m3, and m4.

CFG将程序的控制点显示为节点。 图（a）中的节点显示了每个控制点需要哪些内存块。 例如，表示为节点B1的初始控制点需要加载存储器块m1，m2，m3和m4。

A program can load memory blocks from a storage device such as a hard disk. However, loading and storing memory blocks from such devices is very slow. Hence, most computers contains caches that are used to hold memory blocks and make them available to the program at a much higher speed. The number of blocks that can be held in the cache is determined by the number of cache lines available. Figure (b) above shows that our simplistic program has four cache lines c0, . . . , c3. Memory blocks are loaded into the cache (from the hard disk) based on a given cache replacement policy. We can assume that we have a direct-mapped cache, where memory blocks can be loaded only onto specific cache lines. For instance, figure (b) tells us that blocks m1 and m5 can be loaded only into cache line c0. A cache hit happens when a memory block needed by the program is already in the cache. For instance, if control moves from node B6 to node B8, the program needs the memory block m3, and it is already there. Hence, there is no need to load it from the hard disk. On the other hand, a cache miss happens when a required memory block is not in the cache and needs to be loaded from hard disk onto the cache. For instance, when the program starts up and the cache is empty, the initial node B1 requires us to load all the four memory blocks it needs, resulting in four cache Miss. Analytically, the cache analysis problem boils down to statically determining all possible cache states (and therefore the number of misses in the worst case) at each node using a suitable algorithm. The assignment is divided into the following steps:

程序可以从存储设备（如硬盘）加载内存块。但是，从这些设备加载和存储内存块的速度非常慢。因此，大多数计算机都包含用于保存存储器块的高速缓存，并以更高的速度使其可用于程序。可以在高速缓存中保存的块的数量由可用的高速缓存行的数量决定。上图（b）显示我们的简单程序有四个缓存行c0，...， 。 。 ，c3。内存块将根据给定的缓存替换策略加载到缓存中（从硬盘中）。我们可以假设我们有一个直接映射的缓存，其中内存块只能加载到特定的缓存行。例如，图（b）告诉我们块m1和m5只能加载到高速缓存行c0中。如果程序所需的内存块已经在缓存中，则会发生缓存命中。例如，如果控制从节点B6移动到节点B8，则程序需要存储块m3，并且已经在那里。因此，不需要从硬盘加载它。另一方面，当所需的内存块不在高速缓存中并且需要从硬盘加载到高速缓存上时发生高速缓存未命中。例如，当程序启动并且缓存为空时，

初始节点B1要求我们加载它需要的所有四个内存块，导致四个缓存命中。分析上，高速缓存分析问题归结为使用合适的算法在每个节点处静态确定所有可能的高速缓存状态（并且因此在最坏的情况下确定最坏情况下的错过次数）。

这个任务分为以下几个步骤：

1.Create a Java package called datastructure containing classes for CFG and directmapped cache replacement policy. Create a class TestExample1 which models the sample program and cache provided in the figure on page 1. This question will be marked on the following aspects:

(1) Correct and error free implementation (10 marks).

(2) Ability to scale and flexibly model other programs. For instance, we should be able to add as many cache lines as needed, as easily as possible, and easily add nodes to a program (10 marks).

(3) Quality and organisation of code (5 marks)

翻译：

1.创建一个名为datastructure的Java包，其中包含用于CFG和直接映射缓存替换策略的类。 创建一个类TestExample1，模拟第1页图中提供的示例程序和缓存。这个问题将在以下方面进行标记：

（1）正确和无错执行（10分）。

（2）能够按比例缩放和灵活地模拟其他程序。 例如，我们应该能够尽可能容易地添加尽可能多的缓存行，并轻松地将节点添加到一个程序（10分）。

（3）质量和组织（5分）

1. Design two algorithms to answer the cache analysis problem (20 marks each):
2. Class ExplicitEnumeration enumerates all possible states of the cache at each node in the CFG. Once this exhaustive enumeration is done, it finds the worst number of cache misses for each node in the CFG and returns them.
3. Adapt the algorithm above to create a class CacheLineEnumeration containing an algorithm to enumerate whether a miss can happen in each cache line (and not the whole cache) at each node in the CFG. The algorithm returns the worst case number of cache misses as the number of cache lines that can have a miss along each node in the CFG. (This is not strictly a divide and conquer technique, since the answer returned by this adaptation may not be the same as the result of the first algorithm)

·Bonus: Using any of the techniques taught in the class or via your own research, come up with a third algorithm to compute the worst-case number of cache misses at each node in the CFG (15 marks). (Bonus marks can not be carried over to other assessment items)

翻译：

2.设计两种算法来解答缓存分析问题（每个20个问题）：

（1）Class ExplicitEnumeration列举了CFG中每个节点的缓存的所有可能的状态。一旦完成了这个详尽的枚举，就会发现CFG中每个节点的最高缓存未命中数并返回它们。

（2）调整上述算法，创建一个CacheLineEnumeration类，其中包含一个算法，用于枚举CFG中每个节点的每个缓存行（而不是整个缓存）是否会发生未命中。该算法返回最差情况下的缓存未命中次数作为可能在CFG中的每个节点上未命中的缓存行的数量。 （这不是严格的分而治之的技术，因为这个自适应所返回的答案可能与第一个算法的结果不一样）

·奖金：使用课堂教授的任何技术或通过您自己的研究，提出第三种算法来计算CFG中每个节点（15分）的最高数量的缓存未命中数。