```
1  void garbage()
2  {
3      int *p = (int *)Malloc(15213);
4
5      return; /* Array p is garbage at this point */
6  }
```

因为程序不再需要 p，所以在 garbage 返回前应该释放 p。不幸的是，程序员忘了释放这个块。它在程序的生命周期内都保持为已分配状态，毫无必要地占用着本来可以用来满足后面分配请求的堆空间。

*垃圾收集器*(garbage collector)是一种动态内存分配器，它自动释放程序不再需要的已分配块。这些块被称为*垃圾*(garbage)(因此术语就称之为垃圾收集器)。自动回收堆存储的过程叫做*垃圾收集*(garbage collection)。在一个支持垃圾收集的系统中，应用显式分配堆块，但是从不显示地释放它们。在 C 程序的上下文中，应用调用 malloc，但是从不调用 free。反之，垃圾收集器定期识别垃圾块，并相应地调用 free，将这些块放回到空闲链表中。

垃圾收集可以追溯到 John McCarthy 在 20 世纪 60 年代早期在 MIT 开发的 Lisp 系统。它是诸如 Java、ML、Perl 和 Mathematica 等现代语言系统的一个重要部分，而且它仍然是一个重要而活跃的研究领域。有关文献描述了大量的垃圾收集方法，其数量令人吃惊。我们的讨论局限于 McCarthy 独创的 Mark&Sweep(标记 & 清除)算法，这个算法很有趣，因为它可以建立在已存在的 malloc 包的基础之上，为 C 和 C++ 程序提供垃圾收集。

### 9.10.1　垃圾收集器的基本知识

垃圾收集器将内存视为一张有向可达图(reachability graph)，其形式如图 9-49 所示。该图的节点被分成一组*根节点*(root node)和一组*堆节点*(heap node)。每个堆节点对应于堆中的一个已分配块。有向边 $p \to q$ 意味着块 $p$ 中的某个位置指向块 $q$ 中的某个位置。根节点对应于这样一种不在堆中的位置，它们中包含指向堆中的指针。这些位置可以是寄存器、栈里的变量，或者是虚拟内存中读写数据区域内的全局变量。

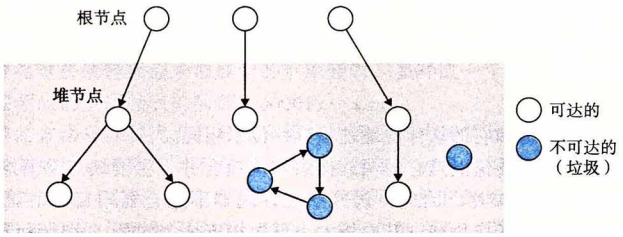


图 9-49　垃圾收集器将内存视为一张有向图

当存在一条从任意根节点出发并到达 $p$ 的有向路径时，我们说节点 $p$ 是*可达的*(reachable)。在任何时刻，不可达节点对应于垃圾，是不能被应用再次使用的。垃圾收集器的角色是维护可达图的某种表示，并通过释放不可达节点且将它们返回给空闲链表，来定期地回收它们。

像 ML 和 Java 这样的语言的垃圾收集器，对应用如何创建和使用指针有很严格的控制，能够维护可达图的一种精确的表示，因此也就能够回收所有垃圾。然而，诸如 C 和