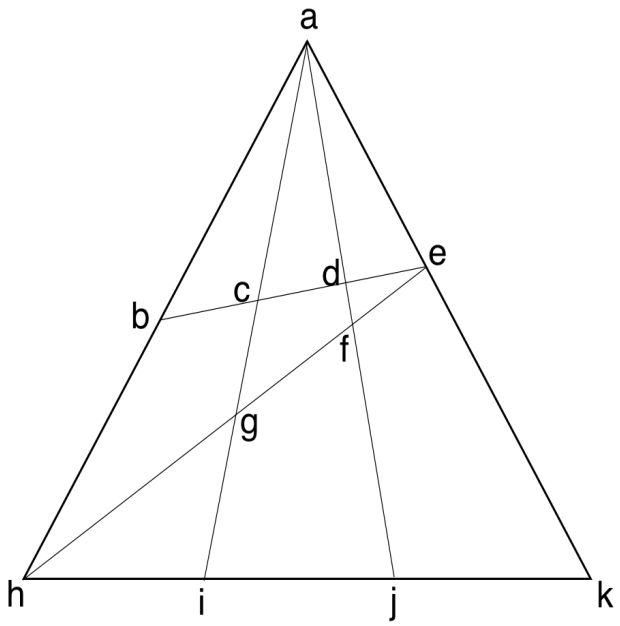
Learning Erlang

practice 1 : count tiangle

题目： 数数如下图形中一共包含多少三角形？



答案是24，你数对了吗？回忆一下你刚才是怎么数的？

这道题如果由人来数，每个人数法各异！但是如果要交给计算机来做，就必须将数的方法描述成计算机可以执行的形式化算法。这种描述方法可以有非常多种，而我们希望找到一套抽象层次高的和领域非常贴切的描述，以降低后续理解、维护成本。

对于该问题，我们首先站在领域角度定义什么是三角形？

```
triangle([A, B, C]) ->
    connected(A, B) andalso
    connected(B, C) andalso
    connected(C, D) andalso
    (not in_same_line(A, B, C)).
```

如上，我们定义了一个三角形就是三个点，两两相连，但是三个点不同时在一条直线上。

对于如上描述，关键是如何定义 `connected` 和 `in_same_line` 。
对于 `connected` ，就是两个点同时在一条直线连接上。那么什么是一条直线连接？
由于我们关注的是点在线上的关系，所以我们定义一条直线为在直线上所有点得集合。
例如对以上例，我们存在直线： `[a, b, h]` , `[a, c, g, i]` 等等。

我们把上图中所有直线用erlang描述出来：

```
lines() ->
    ["abh", "acgi", "adfj", "aek", "bcde", "hgfe", "hijk"].
```

由于我们用单字符表示点，而字符串在erlang中实际就是list，所以我们将一条直线简写为在线上所有点的字符的字符串。

有了对直线的定义，接下来，一个点是否在直线上，那就是元素与集合的属于关系；而两个点是否相连就是集合与集合之间的包含关系。

```
subset([], _S) -> true;
subset([H|T], S) ->
    lists:member(H, S) andalso subset(T, S).

belong(_, []) -> false;
belong(S, [H|T]) -> subset(S, H) orelse belong(S, T).

connected(P1, P2) -> belong([P1, P2], lines()).

in_same_line(P1, P2, P3) -> belong([P1, P2, P3], lines()).
```

可以看到，`connected`的定义是两个点组成的集合属于所有直线的集合的任一个的子集。
而`in_same_line`则是三个点的集合属于所有直线的集合的任一个的子集。
我们在这里将该问题映射到熟悉的集合领域。

在有了对`connected`和`in_same_line`的定以后，我们就可以对`triangle`进行测试了！

```
test() ->
    true = triangle("abc"),
    false = triangle("abh").
```

下来我们来进行数三角形。要能够数三角形，我们需要找到所有三个点的组合，用来验证是否满足`triangle`？将满足`triangle`的进行统计，这样我们就得到了结果！

在这里我们已经有了所有点的集合：

```
Points = "abcdefghijk"
```

为了得到所有3个点的组合，我们实现一个算法，对于集合L，求其N个元素的所有组合的集合。

```
comb(L, 1) -> [[I] || I <- L];
comb(L, N) when length(L) =:= N -> [L];
comb([H|T], N) ->
    [[H|R] || R <- comb(T, N - 1)] ++ comb(T, N).
```

```
Points = "abcdefghijk",
TriplePoints = comb(Points, 3),
```

下面我们实现一个 `count` 方法，用来数满足要求的三角形个数：

```
count(Triple) -> count(Triple, 0).

count([], N) -> N;
count([H|T], N) ->
    case triangle(H) of
        true -> count(T, N + 1);
        false -> count(T, N)
    end.
```

最后可以调用 `run` 测试一下是不是24！

```
run() ->
    Points = "abcdefghijk",
    TriplePoints = comb(Points, 3),
    count(TriplePoints).
```

practice 2 ： Fizz Buzz Whizz

1. 基本完成： erlang语法
2. 分离语义与操作： apply
3. 将apply修改为并行：掌握并行设计
4. 实现pmap，用pmap为每一个apply计算
5. 提前结束进程：熟悉monitor
6. 精确控制apply：实现VM，完成eval
7. 完成compile
8. 并发eval，实现scheduler
9. 虚拟硬件资源，将计算分配到硬件资源上
10. Multi-scheduler，在硬件资源上转移计算