Untitled

我的代码.

```
class Solution(object):
    def canFinish(self, numCourses, prerequisites):
        # build a graph
        # p1: reference value, if pres = [[]] * numCourses
        pres = [[] for _ in range(numCourses)]
        for pair in prerequisites:
            pres[pair[0]].append(pair[1])
        self.pres = pres
        self.ddict = {}

        # keep the visited node
        v = [0] * numCourses
        for i in range(numCourses):
            if v[i] == 0:
                cur_v = []
                if not self.dfs(i, cur_v):
                    return False
                else:
                    for k in cur_v:
                        v[k] = 1
        return True

    def dfs(self, i, cur_v):
        if i in cur_v:
            return False

        cur_v.append(i)
        for j in self.pres[i]:
            tag = self.ddict[j] if j in self.ddict else self.dfs(j, cur_v)
            self.ddict[j] = tag
            if not tag:
                return False
        # p2: err to deal with the dead cycle
        cur_v.pop()
        return True
        # p3 Time Limited

def canFinish(self, numCourses, prerequisites):
    graph = [[] for _ in xrange(numCourses)]
    visit = [0 for _ in xrange(numCourses)]
    for x, y in prerequisites:
        graph[x].append(y)
    def dfs(i):
        if visit[i] == -1:
            return False
        if visit[i] == 1:
            return True
        visit[i] = -1
```

第一次还写错了.

用graph

变量名可以用cache

两个全局变量可以不用直接通过时传参.

多余变量.

完全没有意义

tag变量可以去掉

忘记pop

visit有三种状态太复杂

```
    for j in graph[i]:
        if not dfs(j):
            return False
    visit[i] = 1
    return True
for i in xrange(numCourses):
    if not dfs(i):
        return False
return True
```

code

修改以后的 Code.

```
class Solution(object):
    def canFinish(self, numCourses, prerequisites):
        """
        :type numCourses: int
        :type prerequisites: List[List[int]]
        :rtype: bool
        """
        graph = [[] for _ in range(numCourses)]
        for pair in prerequisites:
            graph[pair[0]].append(pair[1])

        cache = {}
        # keep the visited node
        for i in range(numCourses):
            cur_v = []
            if not self.dfs(i, cur_v, graph, cache):
                return False
        return True

    def dfs(self, i, cur_v, graph, cache):
        if i in cur_v:
            return False

        cur_v.append(i)
        for j in graph[i]:
            cache[j] = cache[j] if j in cache else self.dfs(j, cur_v, graph, cache)
            if not cache[j]:
                return False
        cur_v.pop()

        return True
```