图

Graph

The concept of graph& Relative terminology

图的基本概念及相关术语

The concept of Graph
图Graph的概念

Just like "sheep" is not a separate word in English
就像“羊”在英文中并不是一个单独的词
Chinese "画" has many corresponding words in English, and its meanings are quite different
中文的“画”在英文中有很多对应的单词，其意义大不相同

painting: oil painting painted with a brush
painting：用画刷画的油画

drawing: sketch/line drawing with hard pen
drawing：用硬笔画的素描/线条画

picture: the picture reflected by the real image, such as photo and takepicture
picture：真实形象所反映的画，如照片等，如takepicture

image: paintings from impressions, remote sensing images are used as images, because they come from sensor impressions
image：由印象而来的画，遥感影像做image，因是经过传感器印象而来

The concept of Graph
图Graph的概念

Chinese "picture" has many corresponding words in English, and its meanings are quite different

中文的“图画”在英文中有很多对应的单词，其意义大不相同

figure: the meaning of the outline drawing, the outline of a certain side, so there is a saying of figureout

figure：轮廓图的意思，某个侧面的轮廓，所以有figureout的说法

diagram: abstract conceptual diagrams, such as circuit diagrams, ocean circulation diagrams, and class hierarchy diagrams

diagram：抽象的概念关系图，如电路图、海洋环流图、类层次图

chart: bar chart, pie chart, line chart from digital statistics

chart：由数字统计来的柱状图、饼图、折线图

map: map; plot: a small piece on the map

map：地图；plot：地图上的一小块

graph: focuses on graphs constructed from some basic elements, such as points, line segments, etc.

graph：重在由一些基本元素构造而来的图，如点、线段等

The concept of Graph
图Graph的概念

Graph is a more general structure than a tree, and it is also composed of nodes and edges

图Graph是比树更为一般的结构，也是由节点和边构成

In fact a tree is a graph with special properties

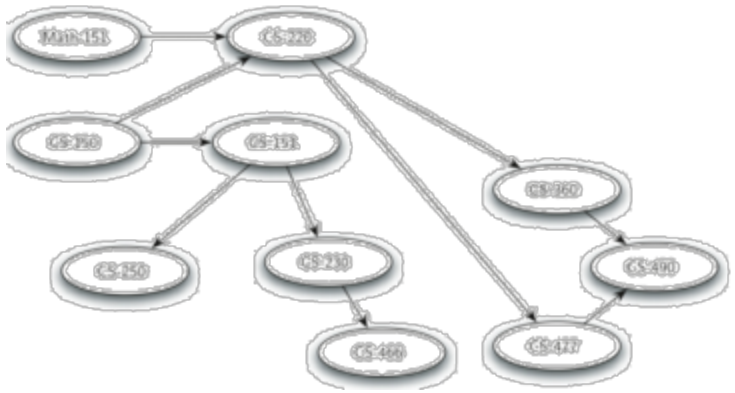
实际上树是一种具有特殊性质的图

Graphs can be used to represent many things in the real world

图可以用来表示现实世界中很多事物

The road transport system, airline routes, internet connection, or the order of precedence for courses at the university

道路交通系统、航班线路、互联网连接、或者是大学中课程的先修次序



The concept of Graph
图Graph的概念

Once we have an accurate description of graph-related problems, standard algorithms for working with graphs can be used to solve problems that seem difficult

一旦我们对图相关问题进行了准确的描述，就可以采用处理图的标准算法来解决那些看起来很艰深的问题

For humans, the recognition patterns of the human brain can easily determine the correlation of different locations on the map;

对于人来说，人脑的识别模式能够轻而易举地判断地图中不同地点的相互关联；

But if a graph is used to represent a map, it can solve many problems that can only be solved by map experts, and even go far beyond it;

但如果用图来表示地图，就可以解决很多地图专家才能解决的问题，甚至远远超越；

The Internet is a complex network connected by thousands of computers, and graph algorithms can also be used to determine the shortest, fastest or most efficient path for communication between computers

互联网是由成千上万的计算机所连接起来的复杂网络，也可以通过图算法来确定计算机之间达成通讯的最短、最快或者最有效的路径。

Beijing public transportation
北京公共交通

The Beijing Metro has 18 operating lines and 268 transfer stations, with a total length of about 527 kilometers.

北京地铁共有18条运营线路，换乘车站则为268座，总长约527千米。

The Beijing public transport system has 1,020 operating lines and nearly 2,000 bus stops.

北京公交系统有1020条运营线路，公交站点近2000个。

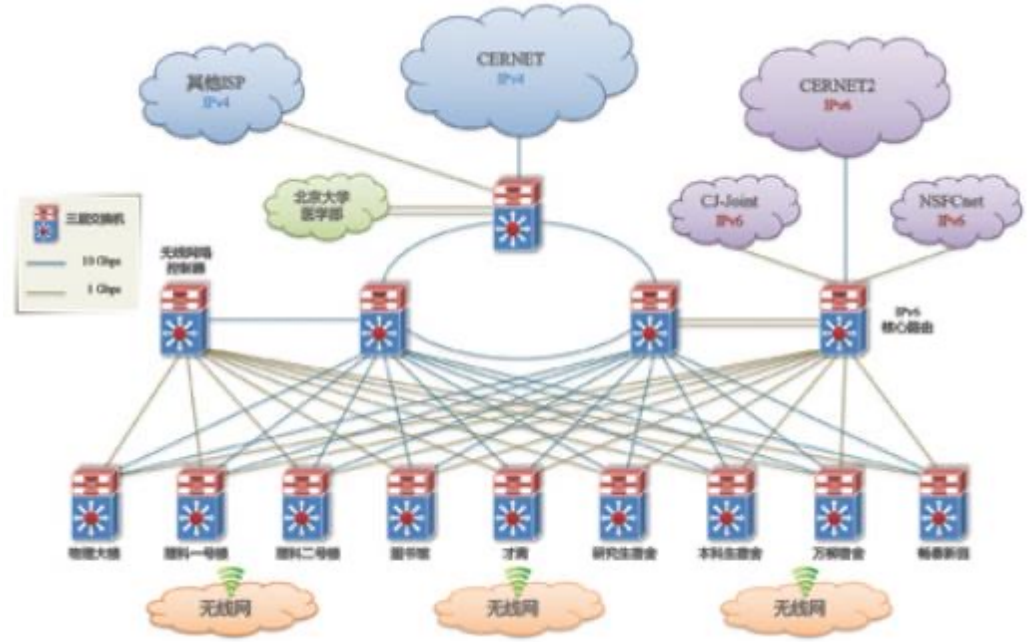
the Internet
互联网

The campus network of Peking University already has more than 100,000 information points

大学校园网已经具有10+万信息点

Through layers of switches, routers are connected together, and routers are connected to each other

通过层层的交换机、路由器连接在一起，路由器之间又相互连接



the Internet
互联网

The Internet is a giant network with tens of billions of information points, and the number of content which Web sites providing has exceeded 1 billion

互联网是一张百亿个信息点的巨型网络，提供内容的Web站点已突破10亿个

There are countless web pages connected by hyperlinks, and Google processes about 10PB of data every day.

由超链接相互连接的网页更是不计其数Google每天处理的数据量约10PB

The human brain can no longer handle the astronomical scale of the network

在天文数字规模的网络面前人脑已经无法处理

Social Networks: The Six Degrees of Separation Theory

社交网络：六度分隔理论

Connect up to 6 people between any two people in the world

世界上任何两个人之间通过最多6个人即可建立联系

The rise of internet social networking brings everyone together

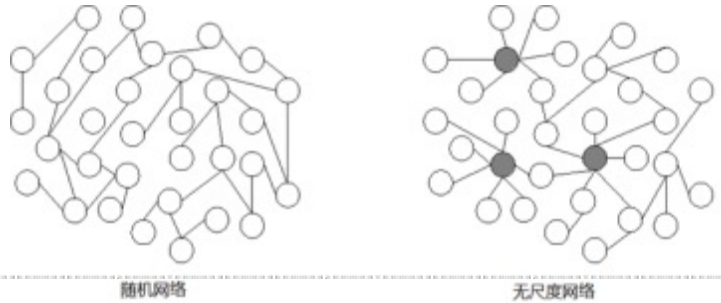
互联网社交网络的兴起将每个人联系到一起

20% of people in society who are good at socializing make 80% of connections

在社会中有20%擅长交往的人，建立了80%的连接

Different from random networks, it ensures the establishment of six degrees of separation, which leads to the study of scale-free networks

区别于随机网络，保证了六度分隔的成立，引出了无尺度网络的研究



Glossary
术语表

Vertex (also called "Node")
顶点Vertex(也称“节点Node”)

It is the basic part of the graph. The vertices have the name identification Key, and can also carry the data item payload.

是图的基本组成部分，顶点具有名称标识Key，也可以携带数据项payload

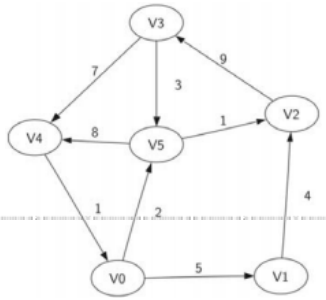
Edge (also called "Arc")
边Edge(也称“弧Arc”)

As a representation of the relationship between 2 vertices, an edge connects two vertices;

作为2个顶点之间关系的表示，边连接两个顶点；

Edges can be undirected or directed, and the corresponding graphs are called "undirected graphs" and "directed graphs"

边可以是无向或者有向的，相应的图称作“无向图”和“有向图”

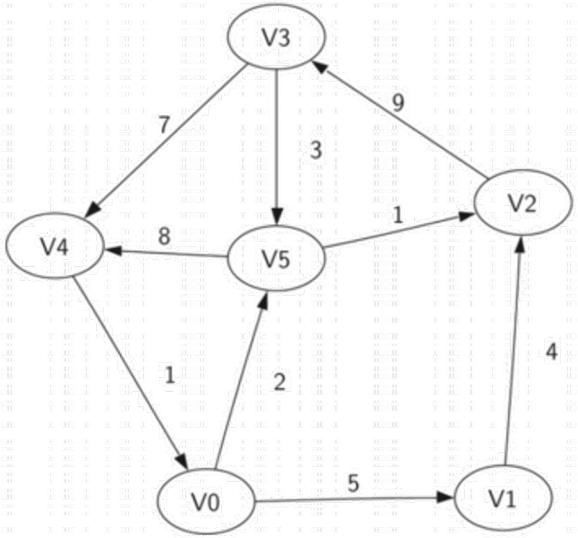


Glossary
术语表

Weight
权重

In order to express the "cost" of going from one vertex to another, edges can be weighted; for example, the "distance", "travel time" and "fare" between two stops in a bus network can all be used as weights.

为了表达从一个顶点到另一个顶点的“代价”，可以给边赋权；例如公交网络中两个站点之间的“距离”、“通行时间”和“票价”都可以作为权重。



Definition of graph
图的定义

A graph G can be defined as G=(V,E)
一个图G可以定义为G=(V,E)

Where V is the set of vertices, E is the set of edges, each edge in E = (v, w), v and w are both vertices in V;

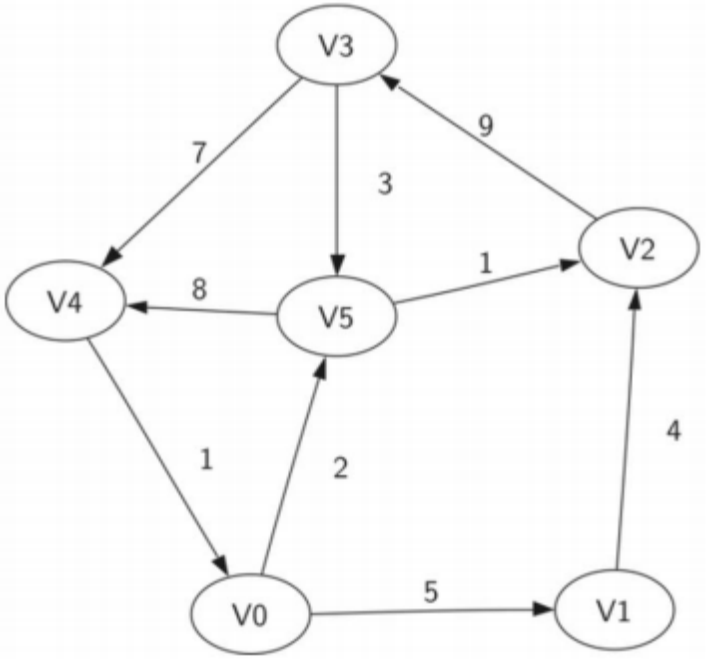
其中V是顶点的集合，E是边的集合，E中的每条边e=(v,w)，v和w都是V中的顶点；

If it is a **weighted graph**, you can add weight components in e

如果是**赋权图**，则可以在e中添加权重分量

Subgraph: a subset of V and E

子图：V和E的子集



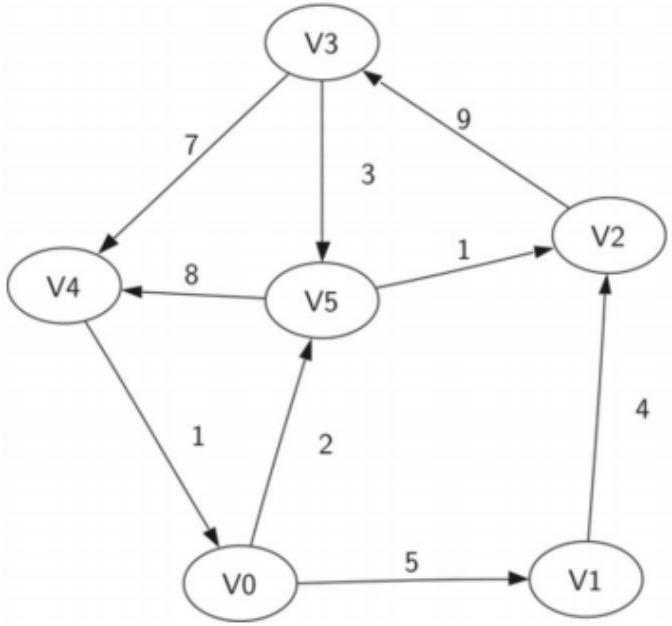
Definition of graph
图的定义

Example of a weighted graph: 6 vertices and 9 edges
赋权图的例子：6个顶点及9条边

Directed weighted graph with integer weights
有向赋权图，权重为整数

$$V = \{V0, V1, V2, V3, V4, V5\}$$

$$E = \left\{ \begin{array}{l} (v0, v1, 5), (v1, v2, 4), (v2, v3, 9), (v3, v4, 7), (v4, v0, 1), \\ (v0, v5, 2), (v5, v4, 8), (v3, v5, 3), (v5, v2, 1) \end{array} \right\}$$



Glossary
术语表

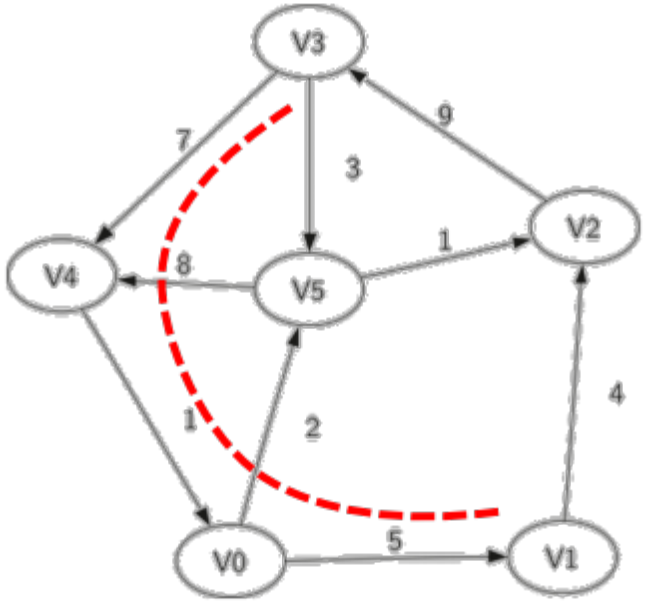
Path
路径

A path in a graph is a sequence of vertices connected in turn by edges;
图中的路径，是由边依次连接起来的顶点序列；

The length of an unweighted path is the number of edges; the length of a weighted path is the sum of all edge weights;
无权路径的长度为边的数量；带权路径的长度为所有边权重的和；

A path as shown below (v3, v4, v0, v1)
如下图的一条路径(v3,v4,v0,v1)

- Its sides are {(v3,v4,7),(v4,v0,1),(v0,v1,5)}
  其边为{(v3,v4,7),(v4,v0,1),(v0,v1,5)}



Glossary
术语表

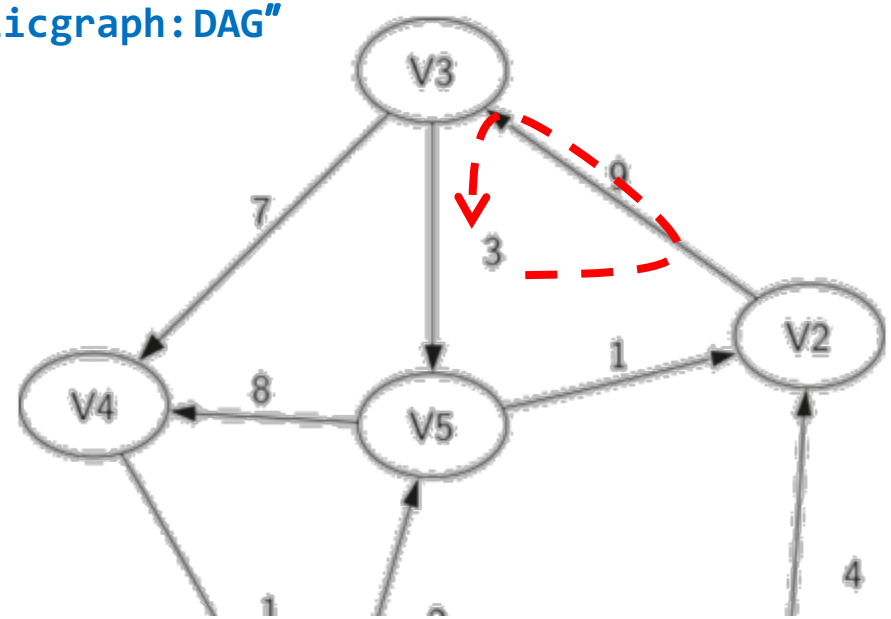
Cycle
圈

A circle is a path with the same head and tail vertices, as shown in the following figure
圈是首尾顶点相同的路径，如下图中

(v5,v2,v3,v5) is a circle
(v5,v2,v3,v5)是一个圈

If there are no cycles in the directed graph, it is called "directed acyclic graph: DAG"
如果有向图中不存在任何圈，则称作“有向无圈directedacyclicgraph:DAG”

Later we can see that if a problem can be represented as a DAG, it can be solved well with graph algorithms.
后面我们可以看到如果一个问题能表示成DAG，就可以用图算法很好地解决。



abstract data type of graph

图抽象数据类型

Abstract data type: ADTGraph
抽象数据类型：ADTGraph

The abstract data type ADTGraph is defined as follows:
抽象数据类型ADTGraph定义如下：

Graph(): Create an empty graph;
Graph()：创建一个空的图；

addVertex(vert): add vertex vert to the graph
addVertex(vert)：将顶点vert加入图中

addEdge(fromVert, toVert): add directed edge
addEdge(fromVert, toVert, weight): add weighted directed edge

getVertex(vKey): Find the vertex named vKey
getVertex(vKey)：查找名称为vKey的顶点

getVertices(): returns a list of all vertices in the graph
getVertices()：返回图中所有顶点列表

in: According to the statement form of vertingraph, return whether the vertex exists in the graph True/False
in：按照vertingraph的语句形式，返回顶点是否存在图中True/False

Abstract data type: ADTGraph
抽象数据类型：ADTGraph

There are two main forms of implementation of ADTGraph:

ADTGraph的实现方法有两种主要形式：

①adjacency matrix

邻接矩阵

②adjacency list

邻接表

Both methods have advantages and disadvantages and need to be selected in different applications

两种方法各有优劣，需要在不同应用中加以选择

Adjacency Matrix
邻接矩阵

Each row and column of the matrix represents a vertex in the graph. If there is an edge between the two vertices, set the respective values.

矩阵的每行和每列都代表图中的顶点，如果两个顶点之间有边相连，设定行列值

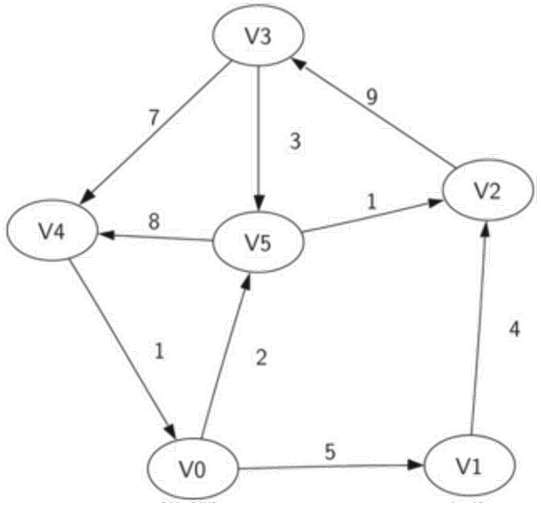
Unweighted edges mark the matrix components as 1, or 0

无权边则将矩阵分量标注为1，或者0

Weighted edges save the weights as matrix component values

带权边则将权重保存为矩阵分量值

	V0	V1	V2	V3	V4	V5
V0		5				2
V1			4			
V2				9		
V3					7	3
V4	1					
V5			1		8	



Adjacency Matrix
邻接矩阵

The advantage of the adjacency matrix implementation method is that it is simple

邻接矩阵实现法的优点是简单

It is easy to get how the vertices are connected

可以很容易得到顶点是如何相连

But it is inefficient if the number of edges in the graph is rare

但如果图中的边数很少则效率低下

becomes a "sparse sparse" matrix

成为“稀疏”矩阵

Wheras most problems correspond to graphs with sparse edges far less than the order of |V|^2

而大多数问题所对应的图都是稀疏的边远远少于|V|^2这个量级

	V0	V1	V2	V3	V4	V5
V0		5				2
V1			4			
V2				9		
V3					7	3
V4	1					
V5			1		8	

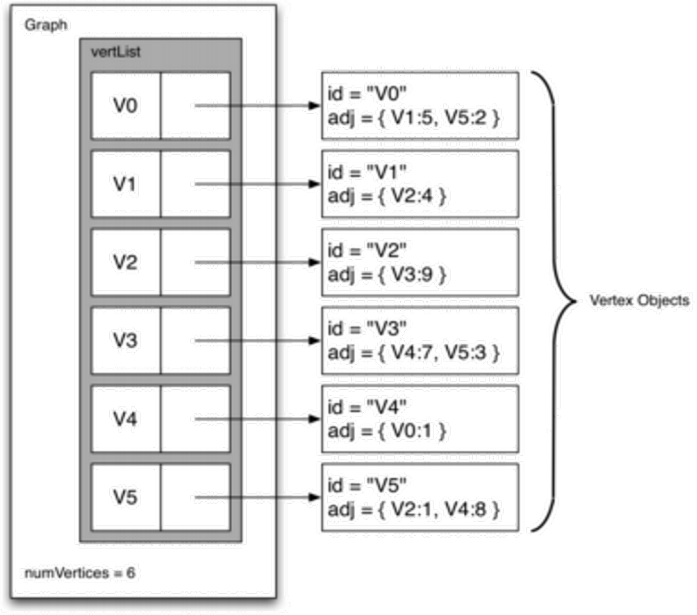
Adjacency List
邻接列表

Adjacency list can be a more efficient implementation of sparse graphs

邻接列表可以成为稀疏图的更高效实现方案

Maintain a master list containing all vertices (masterlist), each vertex in the master list associates a list of all vertices which is connected to itself through edges

维护一个包含所有顶点的主列表(masterlist)，主列表中的每个顶点，再关联一个与自身有边连接的所有顶点的列表



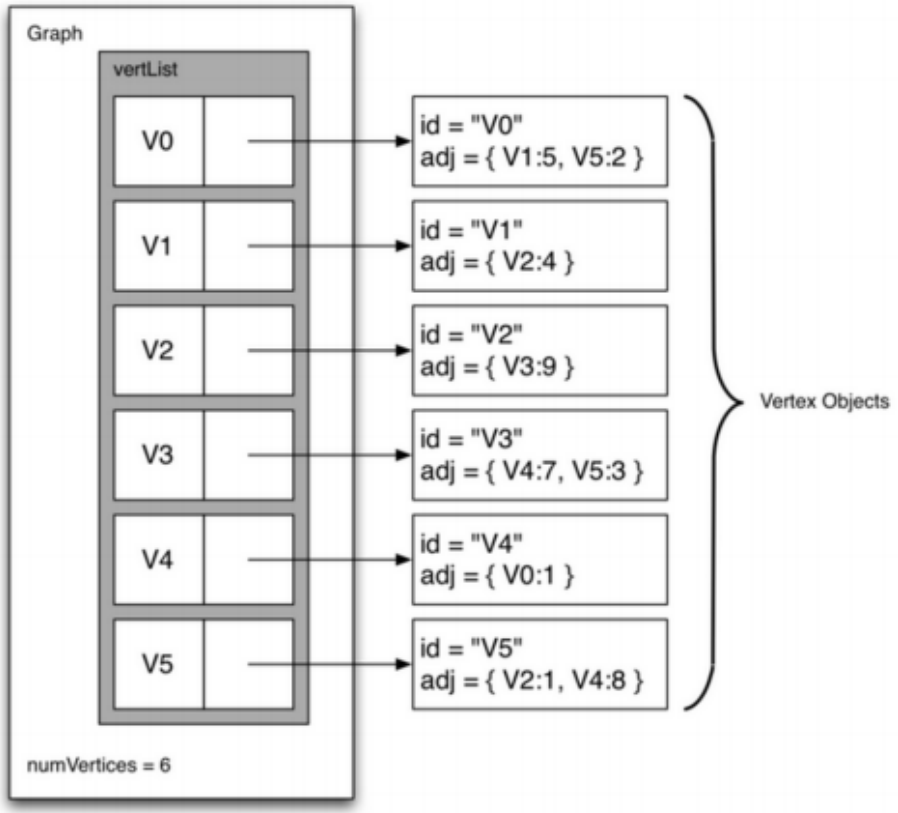
Adjacency List
邻接列表

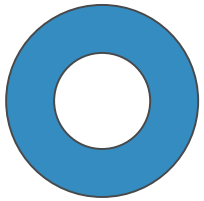
The storage space of the adjacency list method is compact and efficient

邻接列表法的存储空间紧凑高效

It is easy to get all the vertices that the vertices are connected to, as well as the information of the connecting edges

很容易获得顶点所连接的所有顶点，以及连接边的信息





Python implementation of graph abstract data type

图抽象数据类型的Python实现

Implementation of ADTGraph: Examples

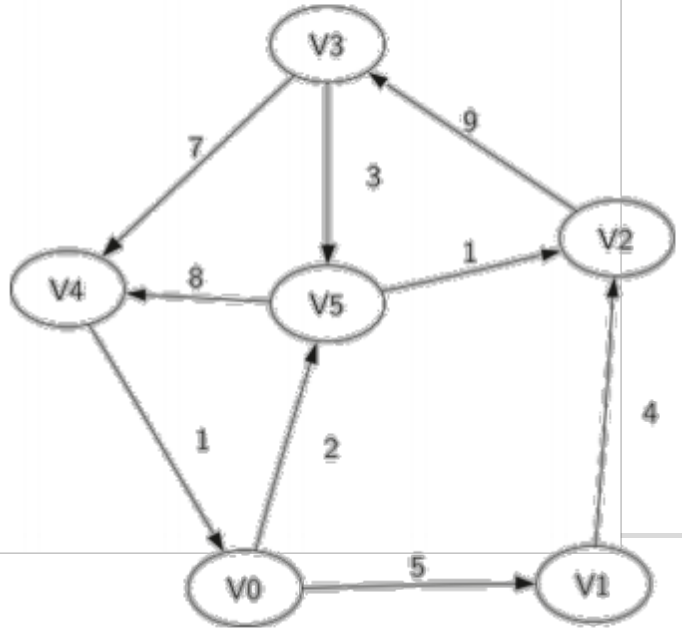
ADTGraph的实现：实例

```
>>> g= Graph()
>>> for i in range(6):
        g.addVertex(i)


0 connectedTo: []
1 connectedTo: []
2 connectedTo: []
3 connectedTo: []
4 connectedTo: []
5 connectedTo: []
>>> print g.vertList
{0: 0 connectedTo: [], 1: 1 connectedTo: [], 2: 2 conne
dTo: [], 4: 4 connectedTo: [], 5: 5 connectedTo: []}
```

Implementation of ADTGraph: Examples

ADTGraph的实现：实例

```
>>> g.addEdge(0,1,5)
>>> g.addEdge(0,5,2)
>>> g.addEdge(1,2,4)
>>> g.addEdge(2,3,9)
>>> g.addEdge(3,4,7)
>>> g.addEdge(3,5,3)
>>> g.addEdge(4,0,1)
>>> g.addEdge(5,4,8)
>>> g.addEdge(5,2,1)
>>> for v in g:
        for w in v.getConnections():
                print "(%s, %s)" % (v.getId(), w.getId())

(0, 5)
(0, 1)
(1, 2)
(2, 3)
(3, 4)
(3, 5)
(4, 0)
(5, 4)
(5, 2)
>>>
```



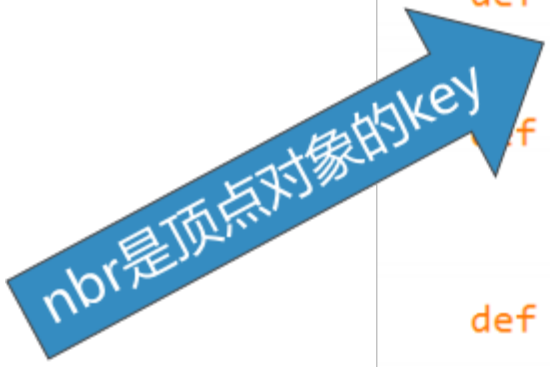
Implementation of ADTGraph: Vertex class

ADTGraph的实现：顶点Vertex类

Vertex contains vertices information, as well as edge information connecting to these vertices

Vertex包含了顶点信息，以及顶点连接边信息

```
class Vertex:
    def __init__(self,key):
        self.id = key
        self.connectedTo = {}

    def addNeighbor(self,nbr,weight=0):
        self.connectedTo[nbr] = weight

    def __str__(self):
        return str(self.id) + ' connectedTo: ' \
               + str([x.id for x in self.connectedTo])

    def getConnections(self):
        return self.connectedTo.keys()

    def getId(self):
        return self.id

    def getWeight(self,nbr):
        return self.connectedTo[nbr]
```



Implementation of ADTGraph: Graph class

ADTGraph的实现：图Graph类

Graph holds the main table containing all vertices

Graph保存了包含所有顶点的主表

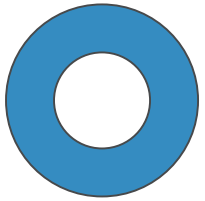

```
class Graph:
    def __init__(self):
        self.vertList = {}
        self.numVertices = 0

    def addVertex(self,key):
        self.numVertices = self.numVertices + 1
        newVertex = Vertex(key)
        self.vertList[key] = newVertex
        return newVertex

    def getVertex(self,n):
        if n in self.vertList:
            return self.vertList[n]
        else:
            return None

    def __contains__(self,n):
        return n in self.vertList
```

Implementation of ADTGraph: Graph class
ADTGraph的实现：图Graph类

```
def __contains__(self,n):
    return n in self.vertList

def addEdge(self,f,t,cost=0):
    if f not in self.vertList:
        nv = self.addVertex(f)
    if t not in self.vertList:
        nv = self.addVertex(t)
    self.vertList[f].addNeighbor(self.vertList[t], cost)

def getVertices(self):
    return self.vertList.keys()

def __iter__(self):
    return iter(self.vertList.values())
```

Non-existing vertices are added first
不存在的顶点先添加

调用起始顶点的方法添加邻接边



Applications of Graphs: The Word Ladder Problem

图的应用：词梯问题

Word Ladder problem
词梯问题

The word game invented by Lewis Carroll, author of "Alice in Wonderland" in 1878

由“爱丽丝漫游奇境”的作者LewisCarroll在1878年所发明的单词游戏

Evolving from one word to another, where the process can go through multiple intermediate words

从一个单词演变到另一个单词，其中的过程可以经过多个中间单词

The requirement is that the difference between two adjacent words can only be 1 letter,

要求是相邻两个单词之间差异只能是1个字母，

For example, FOOL becomes SAGE:

FOOL>>POOL>>POLL>>POLE>>PALE>>SALE>>SAGE

如FOOL变SAGE：FOOL>>POOL>>POLL>>POLE>>PALE>>SALE>>SAGE

WordLadder problem
词梯问题

Our goal is to find the **shortest** sequence of word transformations, and the steps to solve this problem using a graph are as follows:

我们的目标是找到**最短**的单词变换序列,采用图来解决这个问题的步骤如下：

Express the evolution relationship between possible words as a graph, and use "breadth-first search **BFS**" to search for **all** valid paths from the start word to the end word

将可能的单词之间的**演变关**系表达为图,采用“广度优先搜索BFS”，来搜寻从开始单词到结束单词之间的**所有**有效路径

Choose the **fastest** path to the target word

其中**最快到达**目标单词的路径

The Word Ladder Problem: Building a Word Relationship Graph

词梯问题：构建单词关系图

The first is how to put a large set of words into the graph

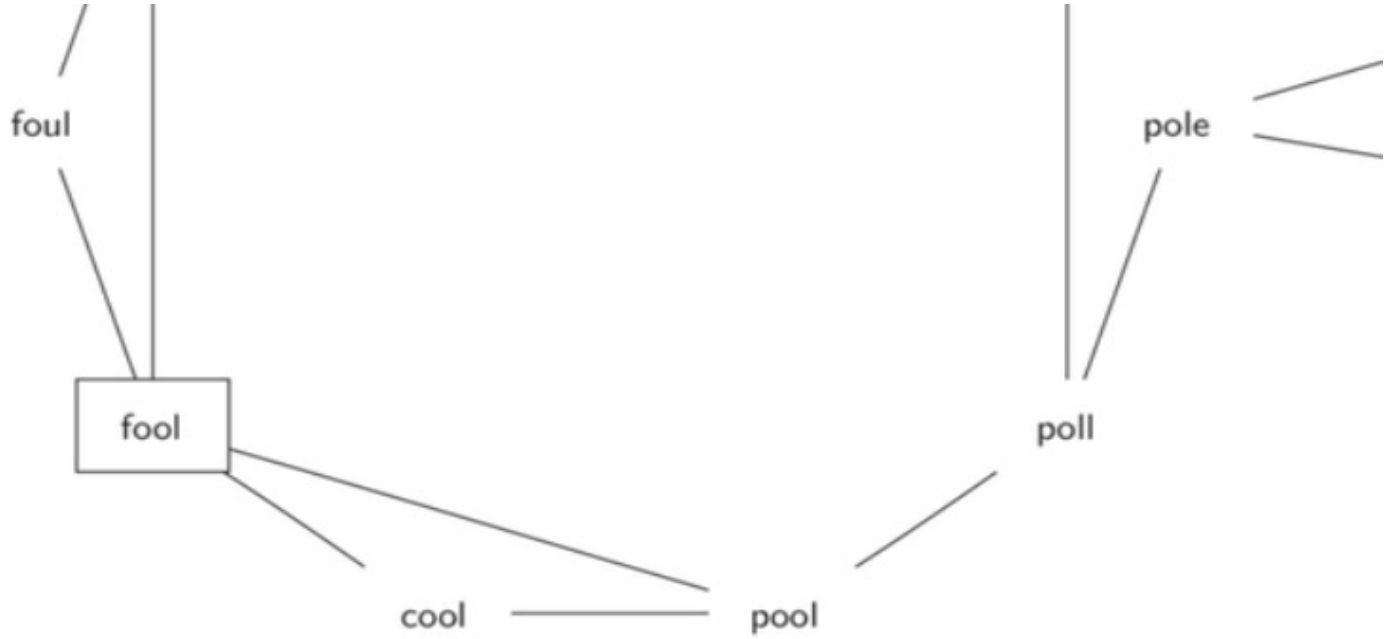
首先是如何将大量的单词集放到图中

- Use the word as the identification *Key* of the vertex

  将单词作为顶点的标识Key

- If two words differ by only 1 letter, put an edge between them

  如果两个单词之间仅相差1个字母，就在它们之间设一条边



The Word Ladder Problem: Building a Word Relationship Graph

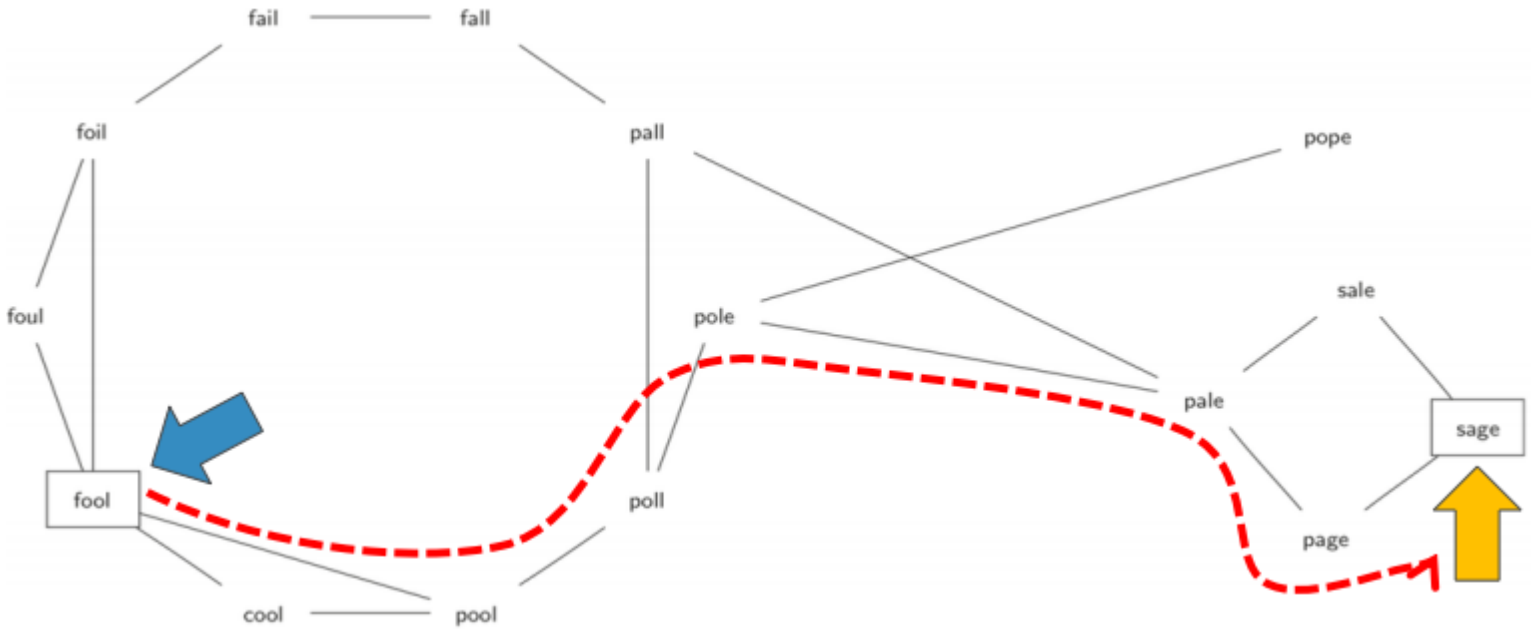
词梯问题：构建单词关系图

The following figure is the word ladder solution from FOOL to SAGE. The graph used is an undirected graph, and the edges have no weights.

下图是从FOOL到SAGE的词梯解，所用的图是无向图，边没有权重

Every path from FOOL to SAGE is a solution

FOOL到SAGE的每条路径都是一个解



The Word Ladder Problem: Building a Word Relationship Graph

词梯问题：构建单词关系图

Word graphs can be constructed by different algorithms (take a 4-letter word list as an example)

单词关系图可以通过不同的算法来构建(以4个字母的单词表为例)

- The first is to add all words to the graph as vertices, and then try to create edges between vertices

  首先是将所有单词作为顶点加入图中，再设法建立顶点之间的边

The most straightforward algorithm for establishing an edge is to compare each vertex (word) with all other words, and create an edge if the difference is only 1 letter

建立边的最直接算法，是对每个顶点(单词)，与其它所有单词进行比较，如果相差仅1个字母，则建立一条边

- Time complexity is O(n^2), requiring more than 26 million comparisons for 5110 words of all 4 letters

  时间复杂度是O(n^2)，对于所有4个字母的5110个单词，需要超过2600万次比较

The Word Ladder Problem: Building a Word Relationship Graph
词梯问题：构建单词关系图

The improved algorithm is to create a large number of buckets, each bucket can store several words

改进的算法是创建大量的桶，每个桶可以存放若干单词

- The bucket mark is a word with one letter removed and the wildcard "_" occupied

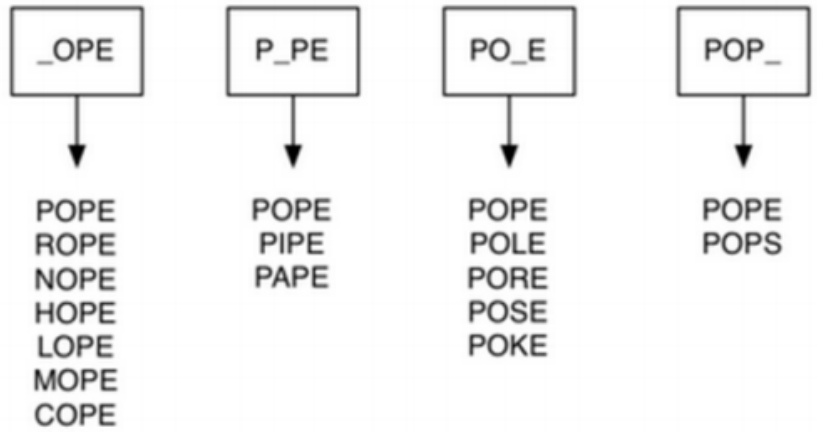
  桶标记是去掉1个字母，通配符“_”占空的单词

All words that match the token are put into this bucket

所有匹配标记的单词都放到这个桶里

- After all words are in place, create edges between words in the same bucket

  所有单词就位后，再在同一个桶的单词之间建立边即可



The word ladder problem: using a dictionary to build buckets

词梯问题：采用字典建立桶

```
def buildGraph(wordFile):
    d = {}
    g = Graph()
    wfile = open(wordFile,'r')
    # create buckets of words that differ by one letter
    for line in wfile:
        word = line[:-1]
        for i in range(len(word)):
            bucket = word[:i] + '_' + word[i+1:]
            if bucket in d:
                d[bucket].append(word)
            else:
                d[bucket] = [word]
    # add vertices and edges for words in the same bucket
    for bucket in d.keys():
        for word1 in d[bucket]:
            for word2 in d[bucket]:
                if word1 != word2:
                    g.addEdge(word1,word2)
    return g
```

4 letter words can belong to 4 buckets
4字母单词可属于4个桶

Create edges between words in the same bucket
同一个桶单词之间建立边

The Word Ladder Problem: Building a Word Relationship Graph

词梯问题：构建单词关系图

The sample data file contains 5,110 4-letter words

样例数据文件包含了5,110个4字母单词

If an adjacency matrix is used to represent this word relationship graph, 26 million matrix elements are required

如果采用邻接矩阵表示这个单词关系图，则需要2,600万个矩阵单元

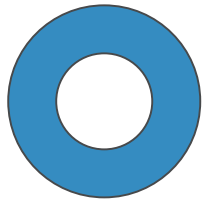
5,110*5,110=26,112,100

The word relation graph has a total of 53,286 edges, which is only 0.2% of the number of matrix cells.

而单词关系图总计有53,286条边，仅仅达到矩阵单元数量的0.2%

The word relation graph is a very sparse graph

单词关系图是一个非常稀疏的图



Implement breadth-first search
实现广度优先搜索

Implement breadth-first search
实现广度优先搜索

After the word relationship graph is established, it is necessary to continue to find the shortest sequence of the word ladder problem in the graph
在单词关系图建立完成以后，需要继续在图中寻找词梯问题的最短序列

Need to use the "**Breadth First Search**" algorithm to search the word relationship graph
需要用到“**广度优先搜索**”算法对单词关系图进行搜索

BFS is one of the simplest algorithms for searching graphs and the basis for some other important graph algorithms
BFS是搜索图的最简单算法之一，也是其它一些重要的图算法的基础

Implement breadth-first search
实现广度优先搜索

Given a graph G, and the starting vertex s to start the search
给定图G，以及开始搜索的起始顶点s

BFS searches for all edges reachable from s
BFS搜索所有从s可到达顶点的边

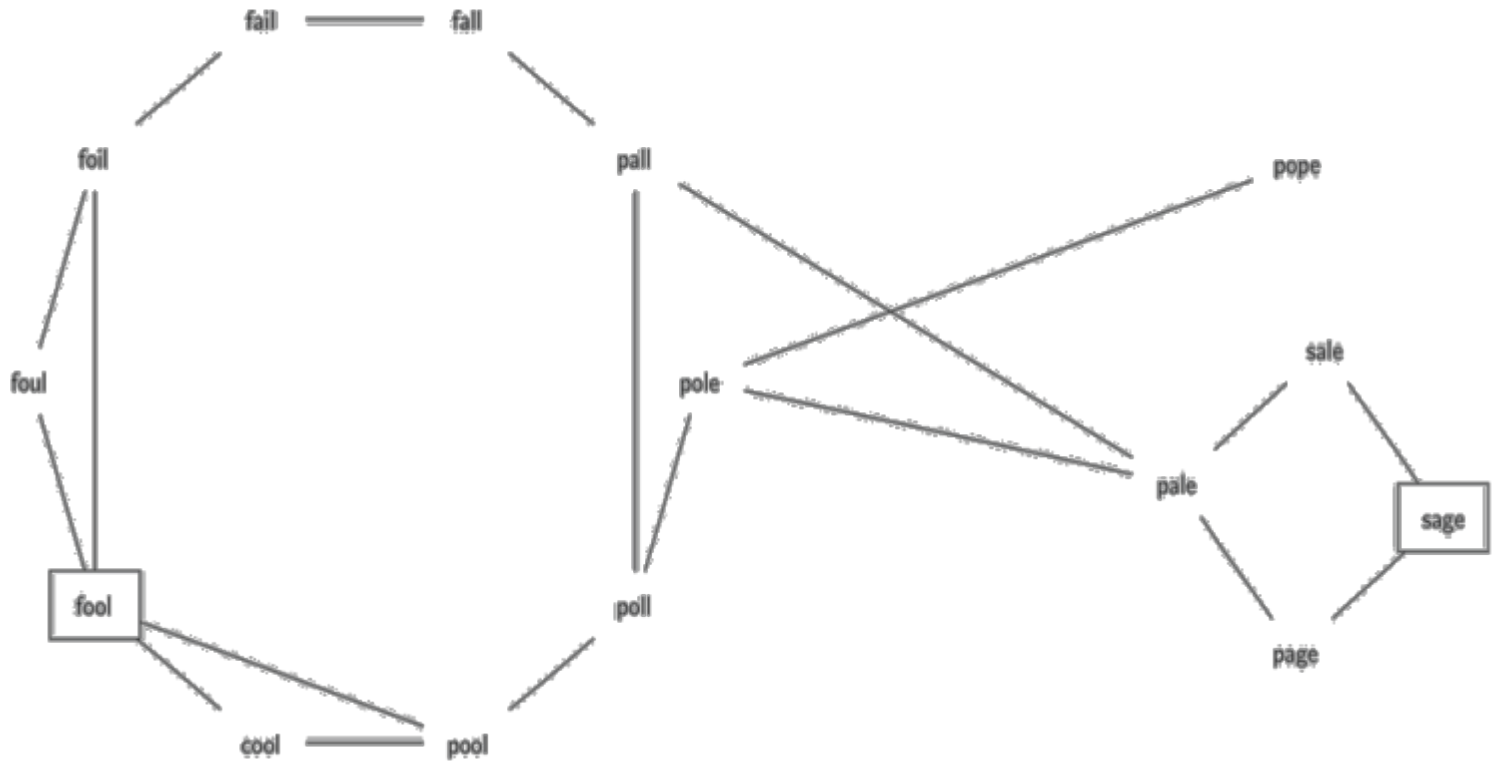
And BFS will find all vertices with distance k before reaching the vertices with further distance k+1
而且在达到更远的距离k+1的顶点之前，BFS会找到全部距离为k的顶点

It can be imagined as the process of building a tree with s as the root, gradually increasing the level from the top down
可以想象为以s为根，构建一棵树的过程，从顶部向下逐步增加层次

Breadth-first search ensures that all sibling nodes are added to the tree before adding levels
广度优先搜索能保证在增加层次之前，添加了所有兄弟节点到树中

BFS algorithm process
BFS算法过程

We start our search with FOOL
我们从FOOL开始搜索



BFS algorithm process
BFS算法过程

In order to track the joining process of vertices and avoid duplication of vertices, add 3 properties to the vertices

为了跟踪顶点的加入过程，并避免重复顶点，要为顶点增加3个属性

①Distance: the length of the path from the starting vertex to this vertex;

距离distance：从起始顶点到此顶点路径长度；

②Predecessor vertex: can be traced back to the starting point;

前驱顶点predecessor：可反向追溯到起点；

③color: identifies whether this vertex has not yet been discovered (white), has been discovered (gray), or has been explored (black)

颜色color：标识了此顶点是尚未发现(白色)、已经发现(灰色)、还是已经完成探索(黑色)

You also need to use a queue *Queue* to arrange the discovered vertices

还需要用一个队列Queue来对已发现的顶点进行排列

Decide which vertex to explore next (the top vertex)

决定下一个要探索的顶点(队首顶点)

BFS algorithm process
BFS算法过程

Starting from the starting vertex *s*, as the newly discovered vertex, it is marked in gray, the distance is 0, the predecessor is *None*, and it is added to the queue. The following content is a loop iteration process:

从起始顶点s开始，作为刚发现的顶点，标注为灰色，距离为0，前驱为None，加入队列，接下来是个循环迭代过程：

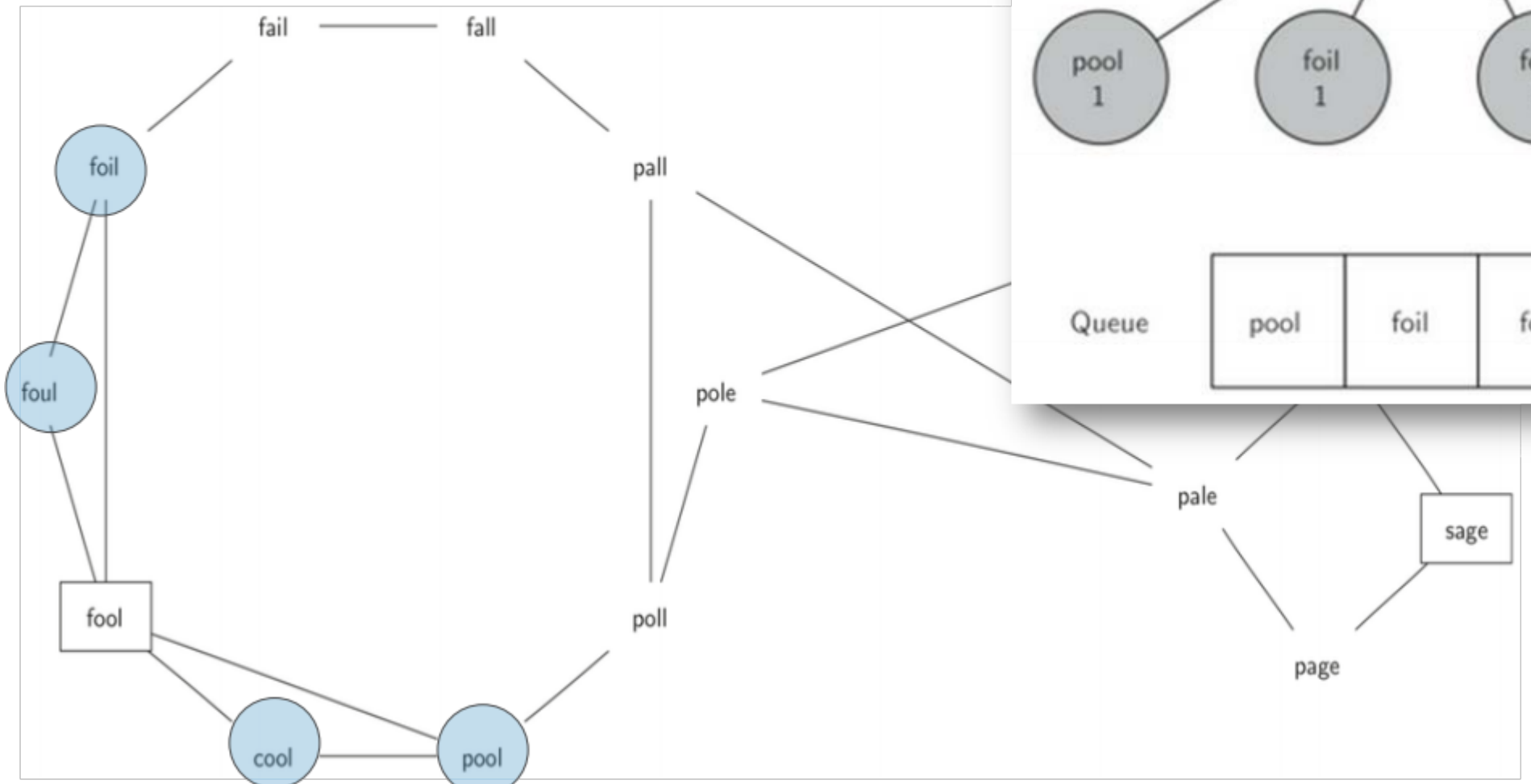
Take a vertex from the head of the team as the current vertex;

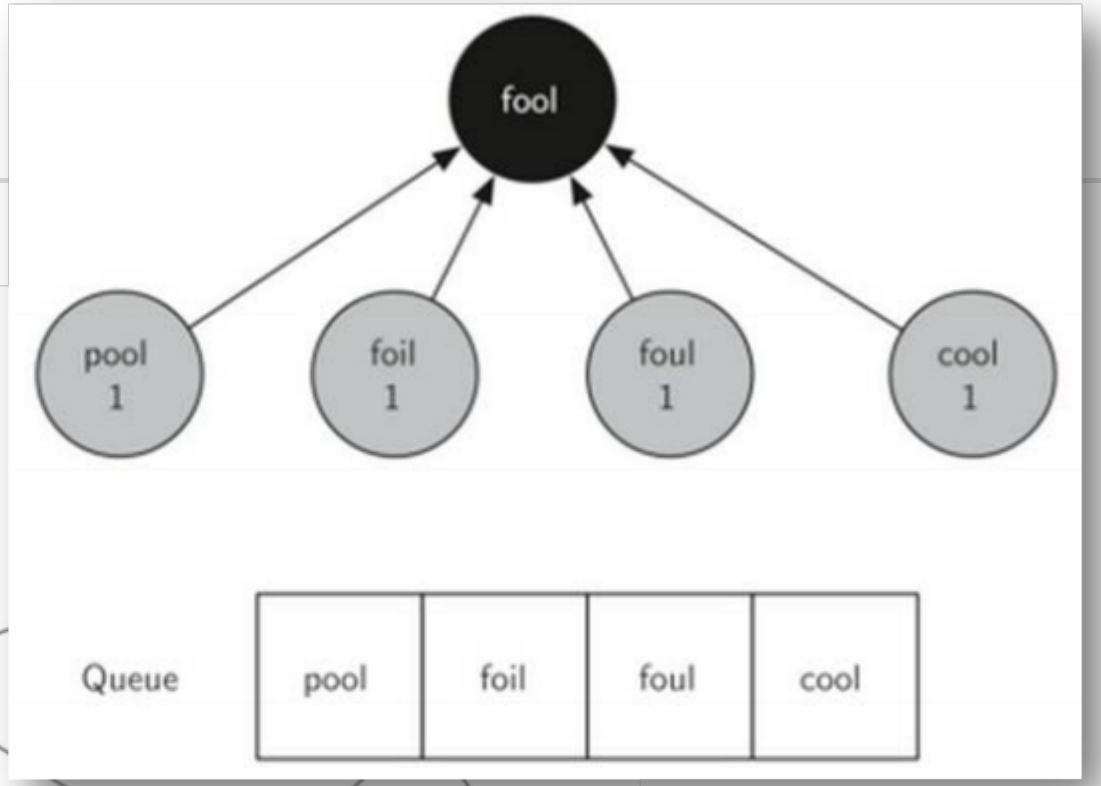
从队首取出一个顶点作为当前顶点；

Traverse the adjacent vertices of the current vertex. If it is an undiscovered white vertex, change its color to gray (found), increase the distance by 1, and the predecessor vertex is the current vertex. After the traversal is completed, the current vertex is set to Black (explored), loop back to the team leader in step 1 to take the current vertex

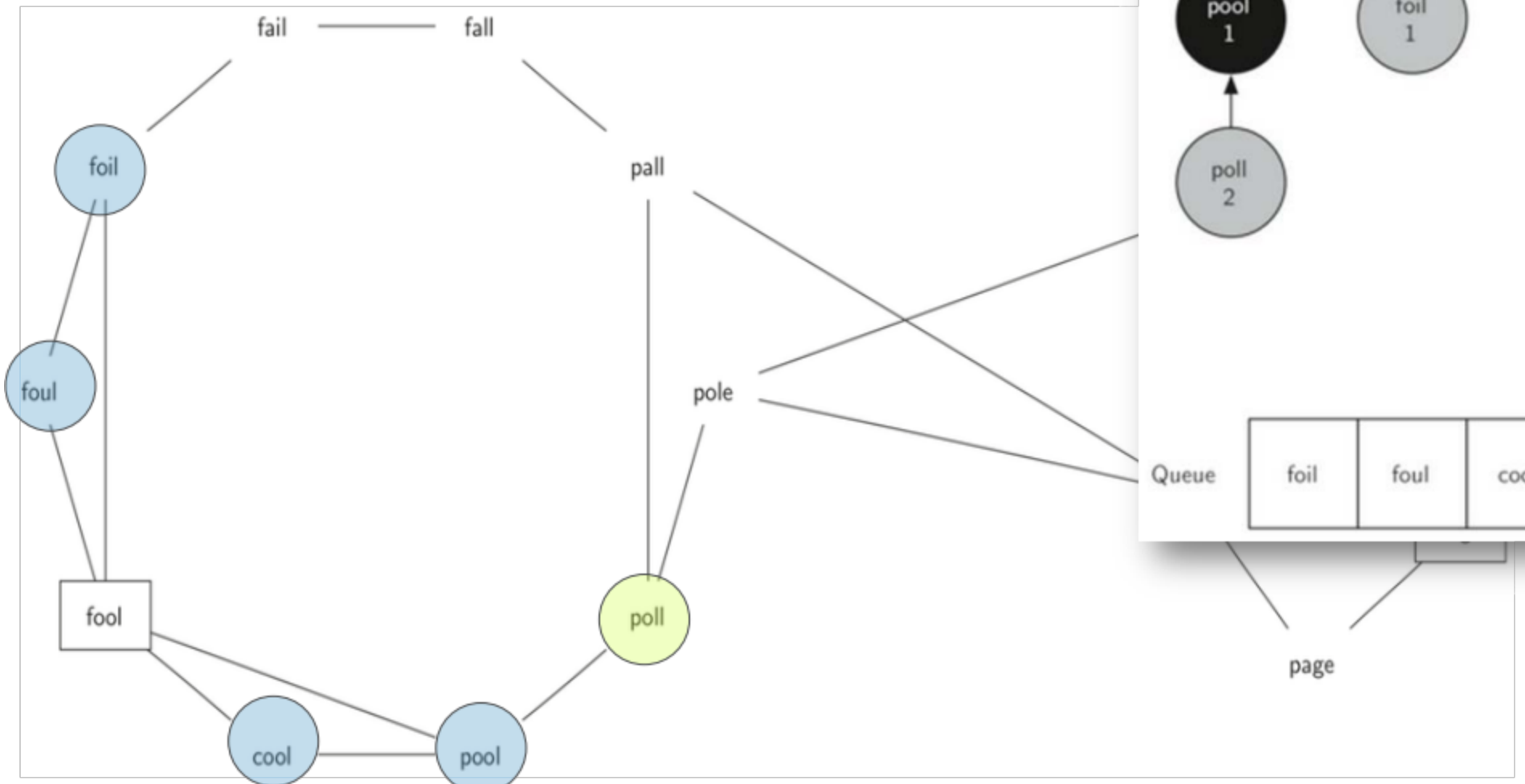
遍历当前顶点的邻接顶点，如果是尚未发现的白色顶点，则将其颜色改为灰色(已发现)，距离增加1，前驱顶点为当前顶点，加入到队列中遍历完成后，将当前顶点设置为黑色(已探索过)，循环回到步骤1的队首取当前顶点

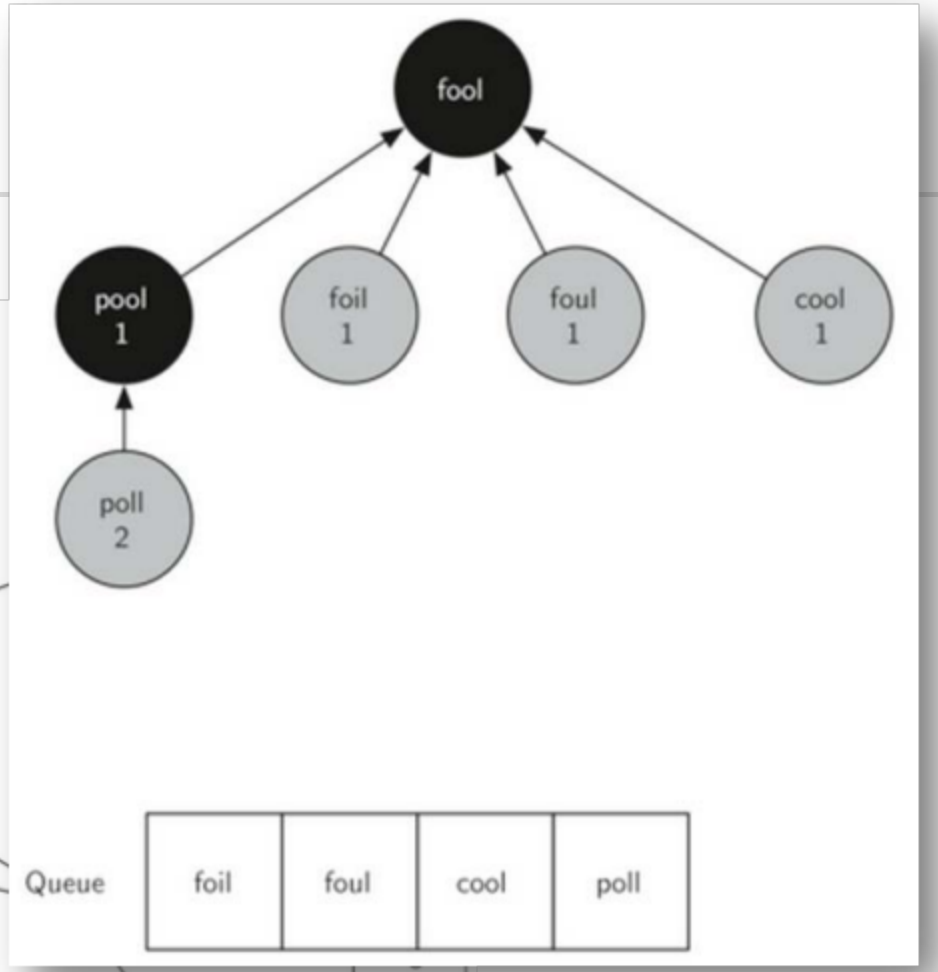
BFS algorithm instance running
BFS算法实例运行



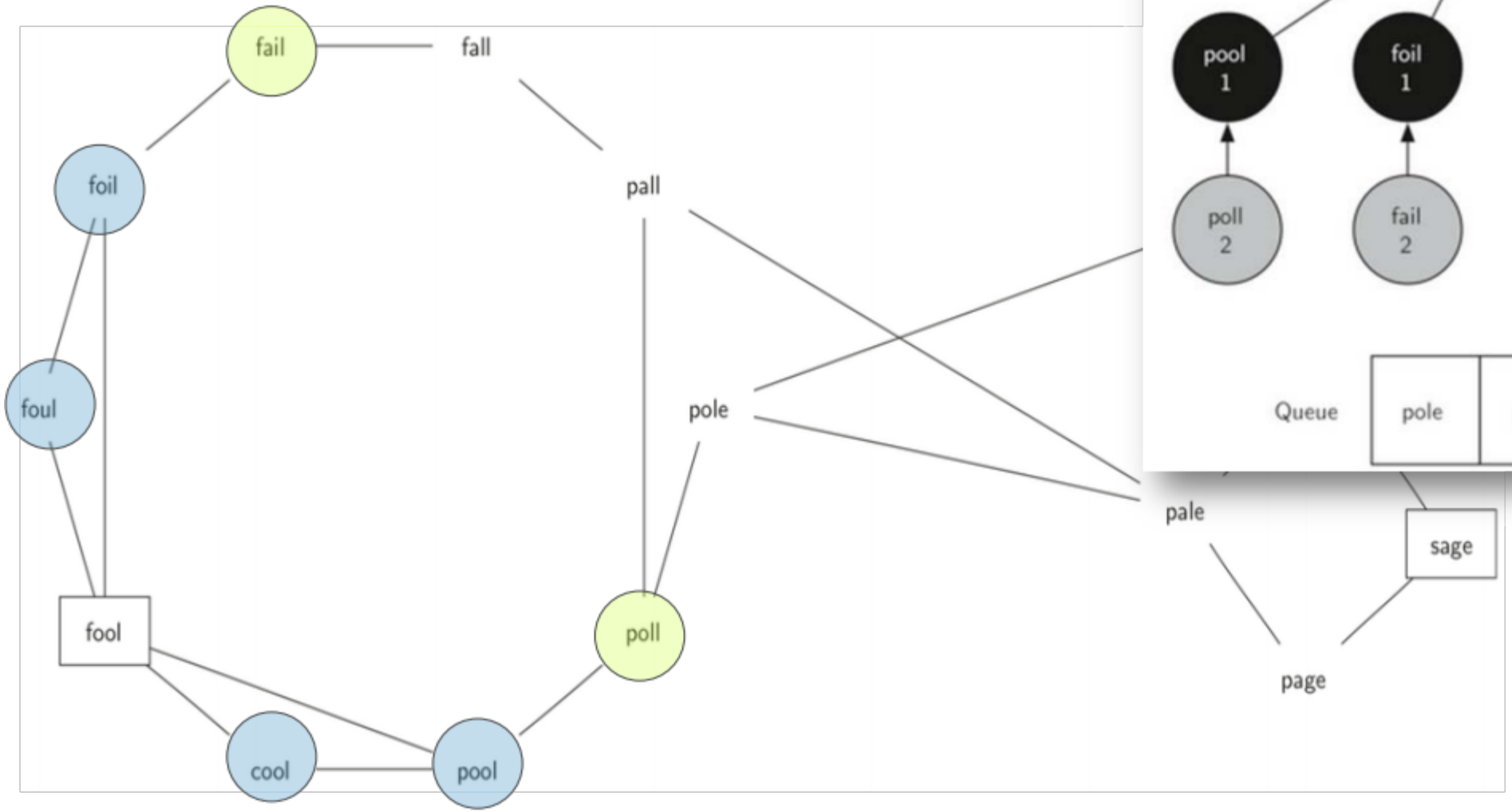


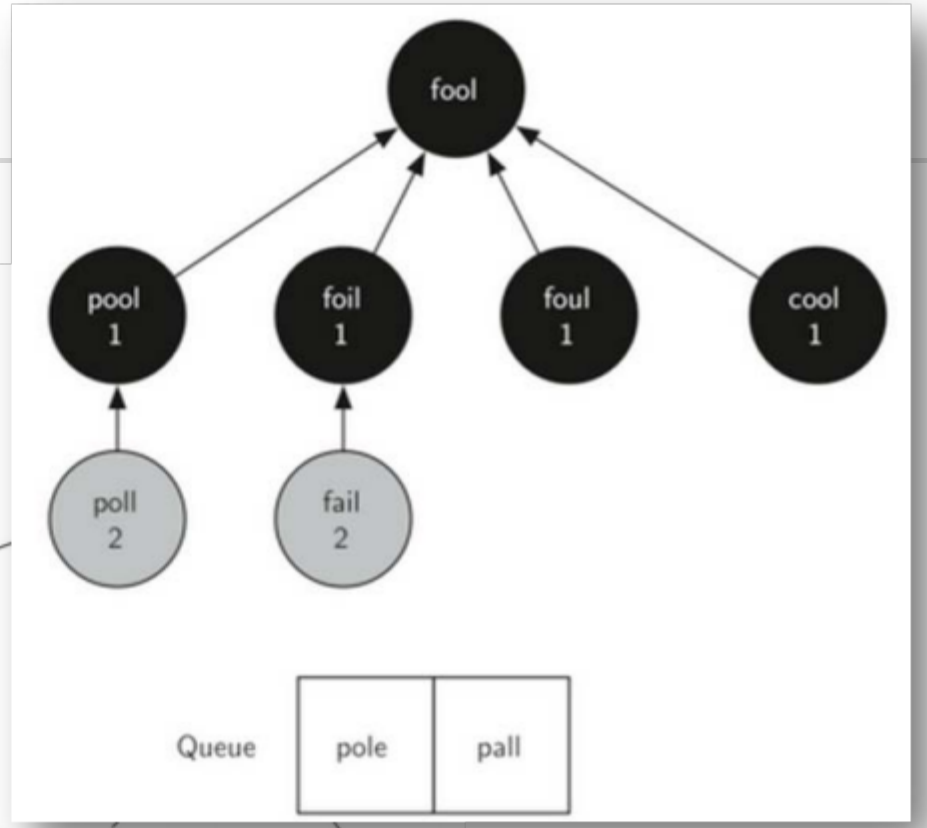
BFS algorithm instance running
BFS算法实例运行



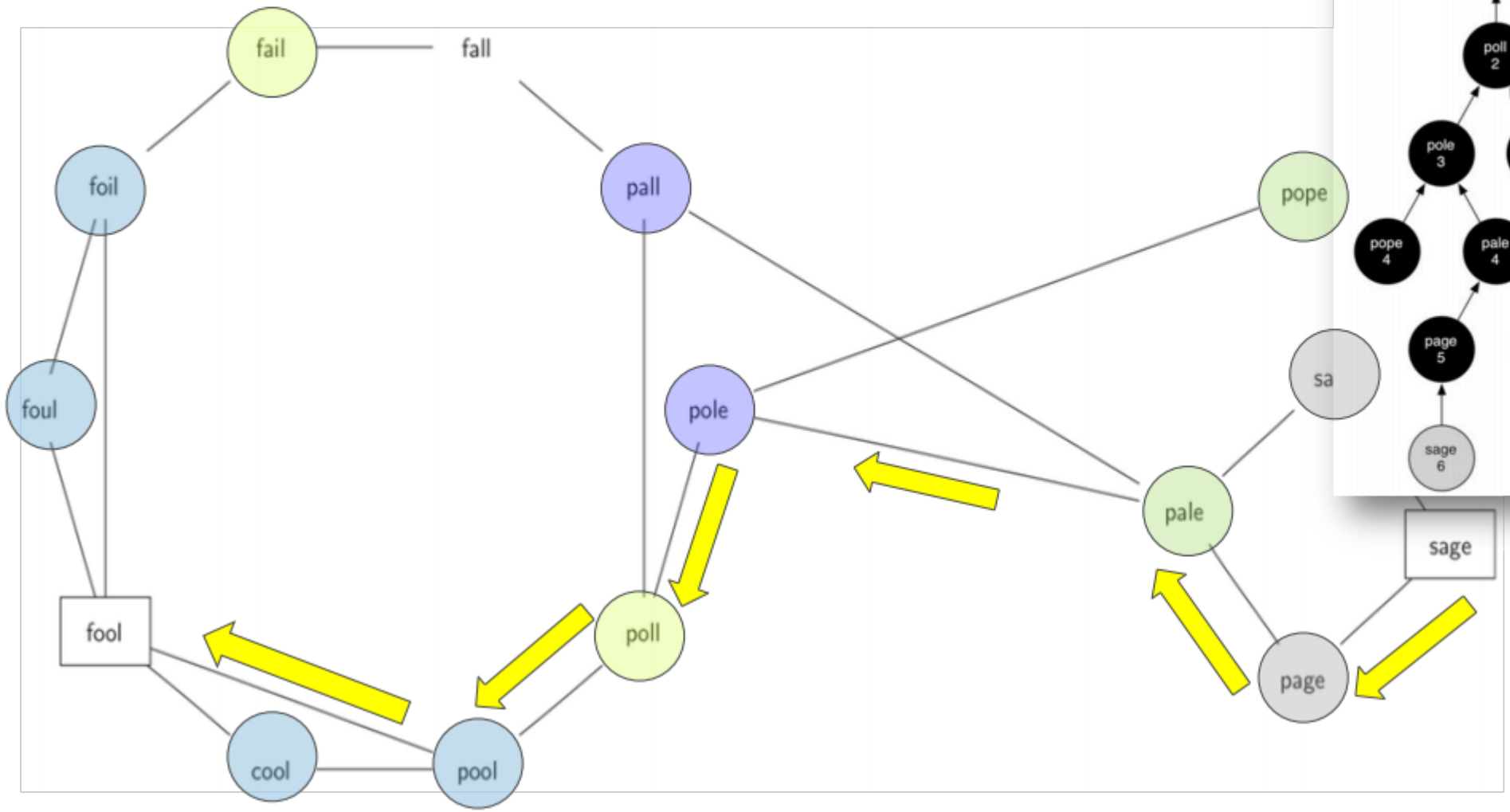


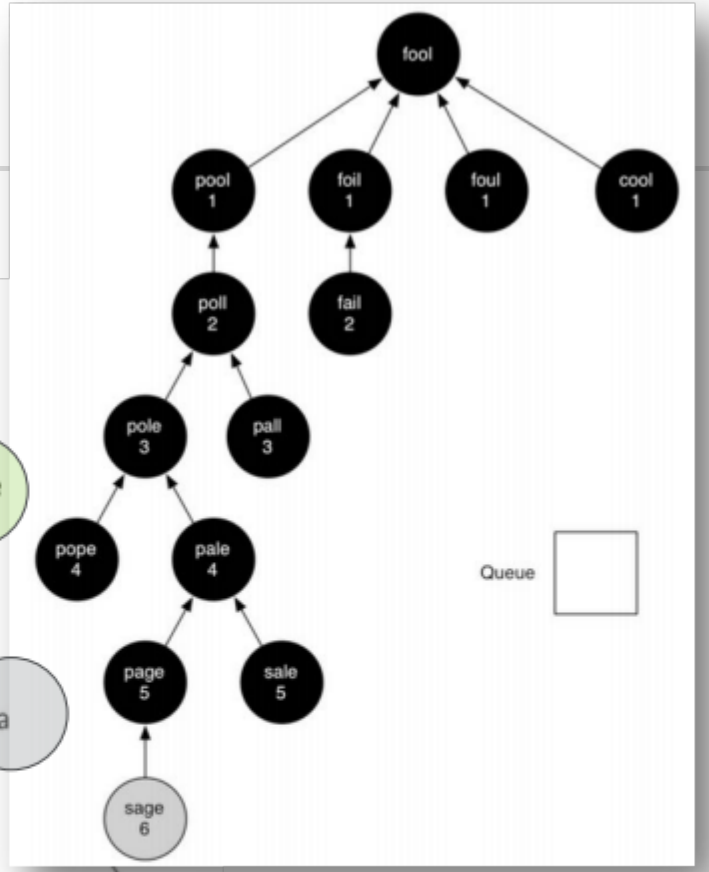
BFS algorithm instance running
BFS算法实例运行





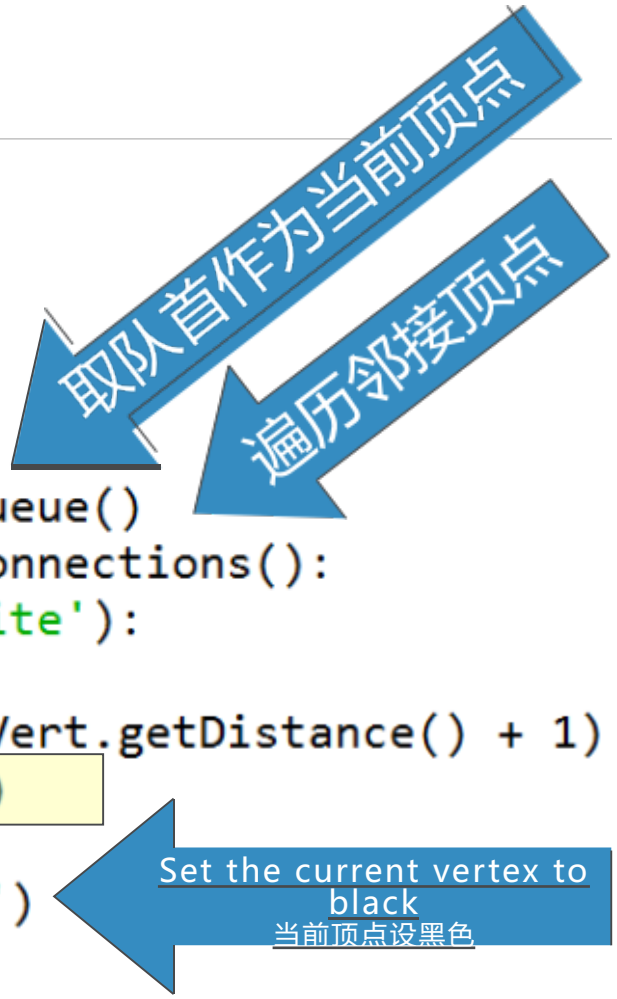
BFS algorithm instance running
BFS算法实例运行





BFS algorithm code
BFS算法代码

```
def bfs(g,start):
  start.setDistance(0)
  start.setPred(None)
  vertQueue = Queue()
  vertQueue.enqueue(start)
  while (vertQueue.size() > 0):
    currentVert = vertQueue.dequeue()
    for nbr in currentVert.getConnections():
      if (nbr.getColor() == 'white'):
        nbr.setColor('gray')
        nbr.setDistance(currentVert.getDistance() + 1)
        nbr.setPred(currentVert)
        vertQueue.enqueue(nbr)
    currentVert.setColor('black')
```



BFS algorithm code
BFS算法代码

After traversing all vertices with FOOL as the starting vertex, and coloring, assigning distances and predecessors to each vertex, a traceback function can be used to determine the shortest word ladder from FOOL to any word vertex!

在以FOOL为起始顶点，遍历了所有顶点，并为每个顶点着色、赋距离和前驱之后，即可以通过一个回途追溯函数来确定FOOL到任何单词顶点的最短词梯！

```
def traverse(y):
    x = y
    while (x.getPred()):
        print(x.getId())
        x = x.getPred()
    print(x.getId())


wordgraph = buildGraph("fourletterwords.txt")

bfs(wordgraph, wordgraph.getVertex('FOOL'))

traverse(wordgraph.getVertex('SAGE'))
```

Analysis of Breadth First Search Algorithm
广度优先搜索算法分析

The body of the BFS algorithm is a nest of two loops
BFS算法主体是两个循环的嵌套

The while loop visits each vertex once, so it is O(|V|), and the for nested in the while, because each edge is only checked once when its starting vertex u is dequeued, and each edge is checked once. A vertex is dequeued at most once, so the edge is checked at most once, which is O(|E|) in total, and the time complexity of BFS is O(|V|+|E|)

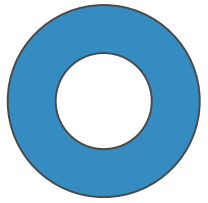
while循环对每个顶点访问一次，所以是O(|V|)，而嵌套在while中的for，由于每条边只有在其起始顶点u出队的时候才会被检查一次，而每个顶点最多出队1次，所以边最多被检查1次，一共是O(|E|)，综合起来BFS的时间复杂度为O(|V|+|E|)

Analysis of Breadth First Search Algorithm
广度优先搜索算法分析

The word ladder problem also includes two part algorithms
词梯问题还包括两个部分算法

①After building the BFS tree, the process of backtracking the vertex to the starting vertex is at most O(|V|)
建立BFS树之后，回溯顶点到起始顶点的过程，最多为O(|V|)

②Creating the word graph also takes time, at most $O(|V|^2)$
创建单词关系图也需要时间，最多为$O(|V|^2)$



Applications of Graphs: The Knight Tour Problem

图的应用：骑士周游问题

knight tour problem
骑士周游问题

On a chess board, a chess piece "horse" (knight), according to the rules of "horse walking in the shape of ‘sun’", starts from a square and travels all the chessboard squares **exactly once.**

在一个国际象棋棋盘上，一个棋子“马”(骑士)，按照“马走日”的规则，从一个格子出发，要走遍所有棋盘格恰好一次。

Call a sequence of moves like this a "travel"

把一个这样的走棋序列称为一次“周游”

35	40	47	44	61	08	15	12
46	43	36	41	14	11	62	09
39	34	45	48	07	60	13	16
50	55	42	37	22	17	10	63
33	38	49	54	59	06	23	18
56	51	28	31	26	21		03
29	32	53	58	05	02	19	24
52	57	30	27	20	25	04	01

knight tour problem
骑士周游问题

On an 8×8 chess board, there are as many as 1.305×10^{35} qualified "tours", and there are even more tours that fail in the process of making moves.

在8×8的国际象棋棋盘上，合格的"周游"数量有1.305×10^{35}这么多，走棋过程中失败的周游就更多了

Using a graph search algorithm is one of the easiest solutions to understand and to program to solve the knight travel problem.

采用图搜索算法，是解决骑士周游问题最容易理解和编程的方案之一

The solution is still divided into two steps:

解决方案还是分为两步：

①First, the legal moving sequence is represented as a graph

首先将合法走棋次序表示为一个图

②Use the graph search algorithm to search for a path of length (row×column-1) that contains each vertex exactly once

采用图搜索算法搜寻一个长度为(行×列-1)的路径，路径上包含每个顶点恰一次

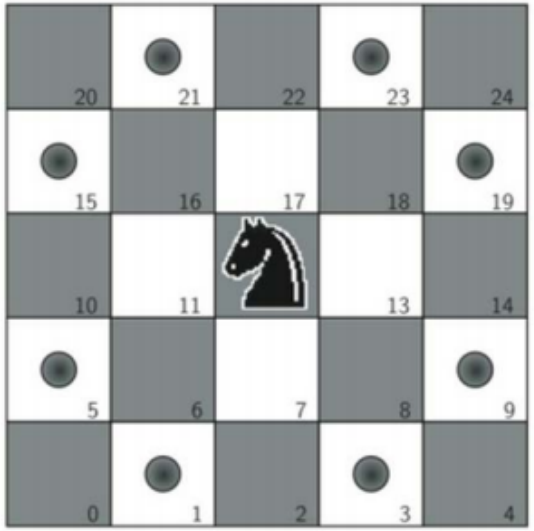
Build a knight tour map
构建骑士周游图

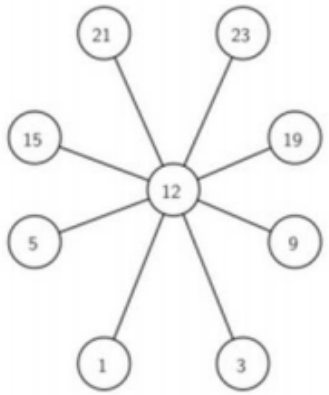
The idea of constructing the chessboard and the moves as a graph
将棋盘和走棋步骤构建为图的思路

Checkerboard as vertex
将棋盘格作为顶点

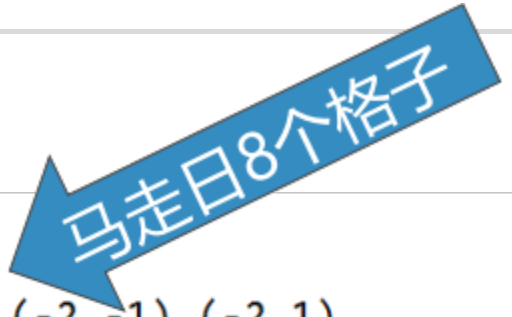
According to the "horse move in the shape of sun" rule, the step of moving chess is used as connecting edges
按照“马走日”规则,走棋步骤作为连接边

Build a graph of the checkerboard relationships that can be reached by all legal moves of each checkerboard
建立每一个棋盘格的所有合法走棋步骤能够到达的棋盘格关系图





legal move position function
合法走棋位置函数




```
def genLegalMoves(x,y,bdSize):
    newMoves = []
    moveOffsets = [(-1,-2),(-1,2),(-2,-1),(-2,1),
                   ( 1,-2),( 1,2),( 2,-1),( 2,1)]
    for i in moveOffsets:
        newX = x + i[0]
        newY = y + i[1]
        if legalCoord(newX,bdSize) and legalCoord(newY,bdSize):
            newMoves.append((newX,newY))
    return newMoves

def legalCoord(x,bdSize):
    if x >= 0 and x < bdSize:
        return True
    else:
        return False
```

Build a chess-moving relationship graph
构建走棋关系图

```
def knightGraph(bdSize):
    ktGraph = Graph()
    for row in range(bdSize):
        for col in range(bdSize):
            nodeId = posToNodeId(row,col,bdSize)
            newPositions = genLegalMoves(row,col,bdSize)
            for e in newPositions:
                nid = posToNodeId(e[0],e[1],bdSize)
                ktGraph.addEdge(nodeId,nid)
    return ktGraph

def posToNodeId(row,col,bdSize):
    return row*bdSize+col
```

遍历每个格子

single legal move
单步合法走棋

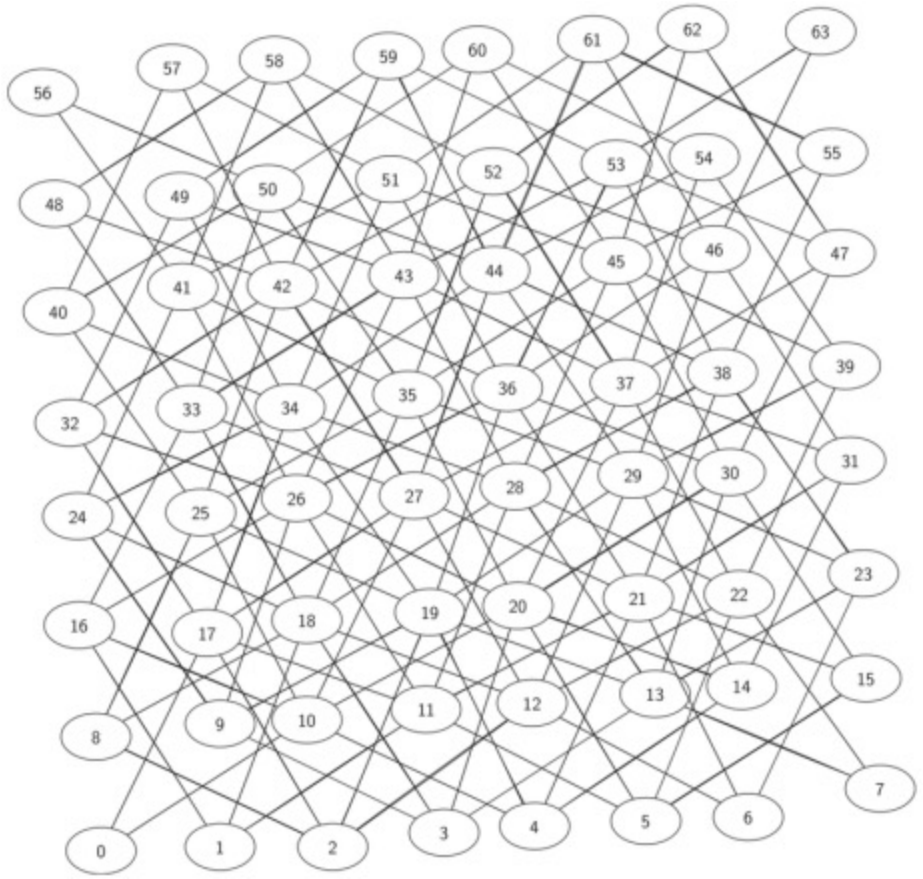
Add edges and vertices
添加边及顶点

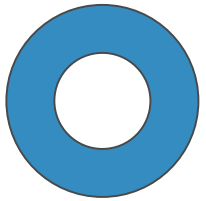
Knight's Tour Map: A Map Generated by an 8×8 Chessboard

骑士周游图：8×8棋盘生成的图

It has 336 edges, which is only 8.2% compared to the fully connected 4096 edges, which is still a sparse graph

具有336条边，相比起全连接的4096条边，仅8.2%，还是稀疏图





Implementation of the Algorithm for the Knight's Tour Problem

骑士周游问题算法实现

Implementation of the knight tour algorithm
骑士周游算法实现

The graph search algorithm used to solve the knight tour problem is DepthFirstSearch

用于解决骑士周游问题的图搜索算法是深度优先搜索(DepthFirstSearch)

Compared with the aforementioned breadth-first search, it has the characteristics of building a search tree layer by layer.

相比前述的广度优先搜索，其具有逐层建立搜索树的特点

Depth-first search is to search **as deep as possible** down a single branch of the tree

深度优先搜索是沿着树的单支尽量深入向下搜索

If the solution to the problem is not found to the extent that it cannot continue, go back to the previous layer and search for the next one.

如果到无法继续的程度还未找到问题解，就回溯上一层再搜索下一支

Implementation of the knight tour algorithm
骑士周游算法实现

Two implementation algorithms of DFS will be introduced later.

后面会介绍DFS的两个实现算法

A DFS algorithm for solving the knight-tour problem, characterized by the fact that each vertex is visited only once

一个DFS算法用于解决骑士周游问题，其特点是每个顶点仅访问一次

Another DFS algorithm is more general and allows vertices to be visited repeatedly so it can be used as the basis for other graph algorithms

另一个DFS算法更为通用，允许顶点被重复访问，可作为其它图算法的基础

Implementation of the knight travel algorithm
骑士周游算法实现

The key idea of depth-first search to solve the knight tour
深度优先搜索解决骑士周游的关键思路

If you can't continue to search further along a single branch (all legal moves have been walked)

如果沿着单支深入搜索到无法继续(所有合法移动都已经被走过了)时

If the path length has not reached the predetermined value (63 for an 8×8 chessboard), then clear the color mark, return to the previous layer, and switch to another branch to continue the search.

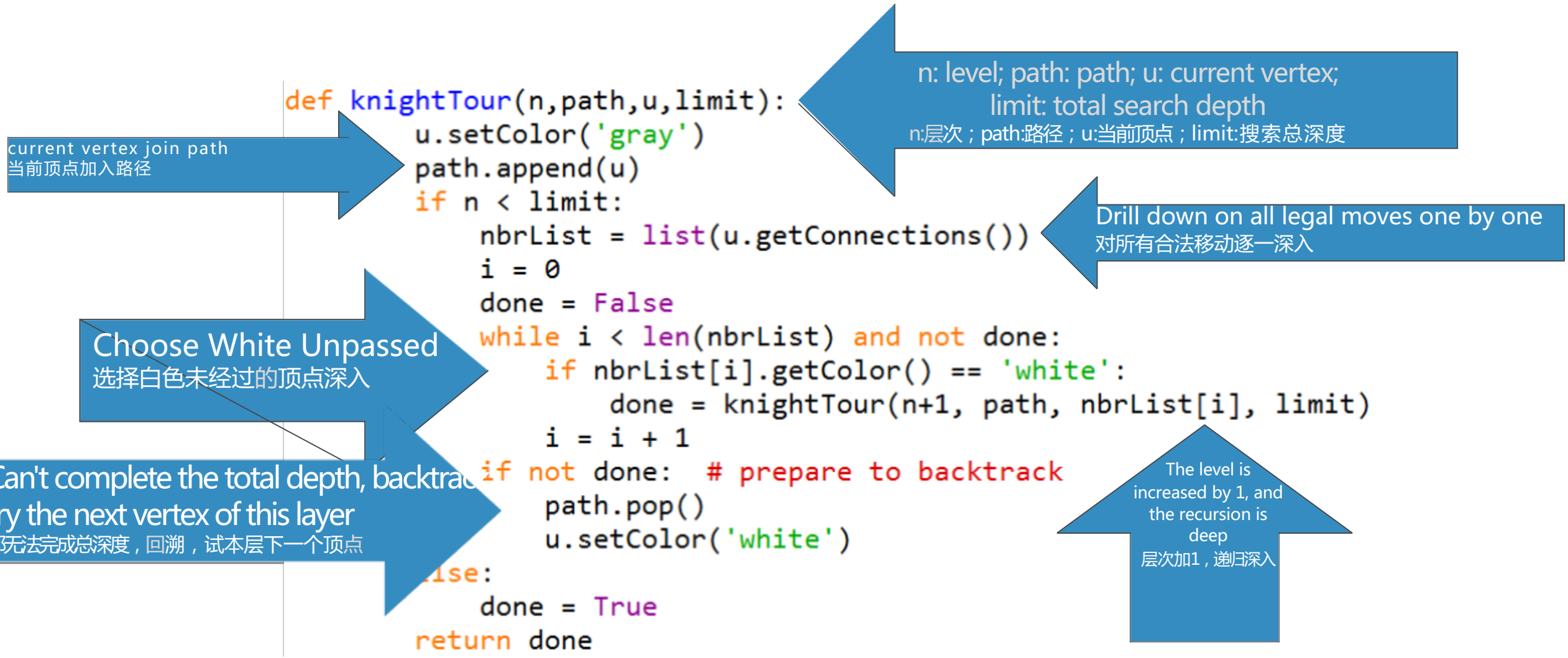
路径长度还没有达到预定值(8×8棋盘为63)那么就清除颜色标记，返回到上一层，换一个分支继续深入搜索

Introduce a stack to record the path
引入一个栈来记录路径

And implement the backtracking operation that returns to the previous layer

并实施返回上一层的回溯操作

knight tour algorithm code
骑士周游算法代码



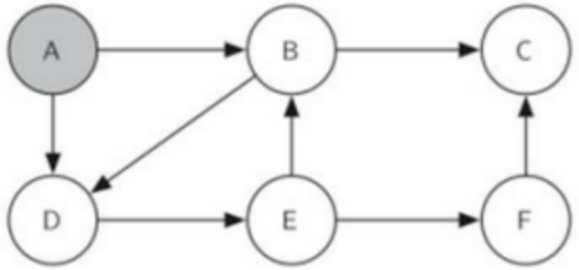


Figure 3: Start with node A

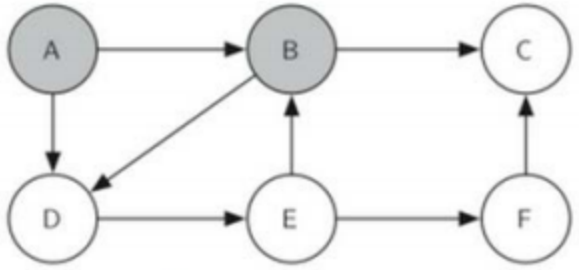


Figure 4: Explore B

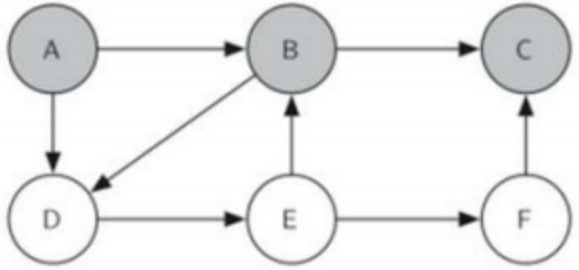


Figure 5: Node C is a dead end

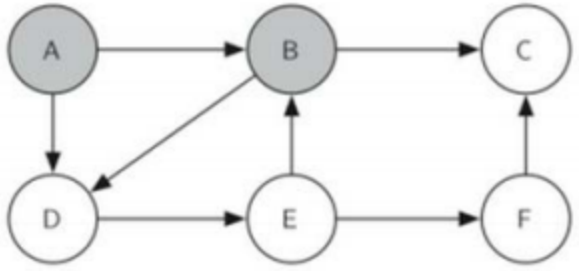


Figure 6: Backtrack to B

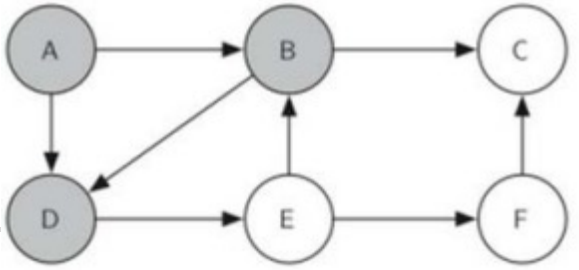


Figure 7: Explore D

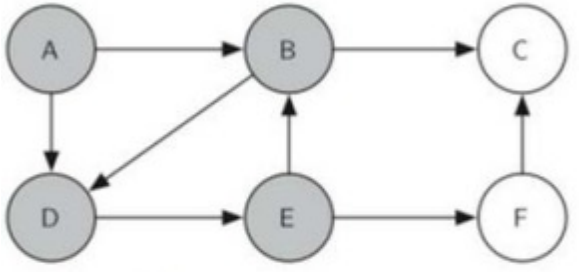


Figure 8: Explore E

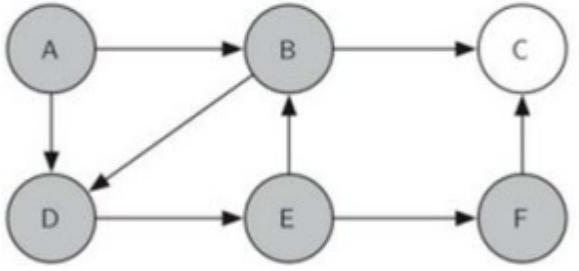


Figure 9: Explore F

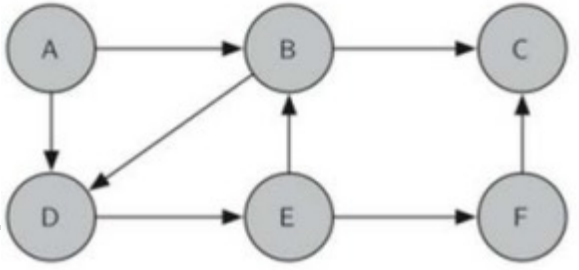
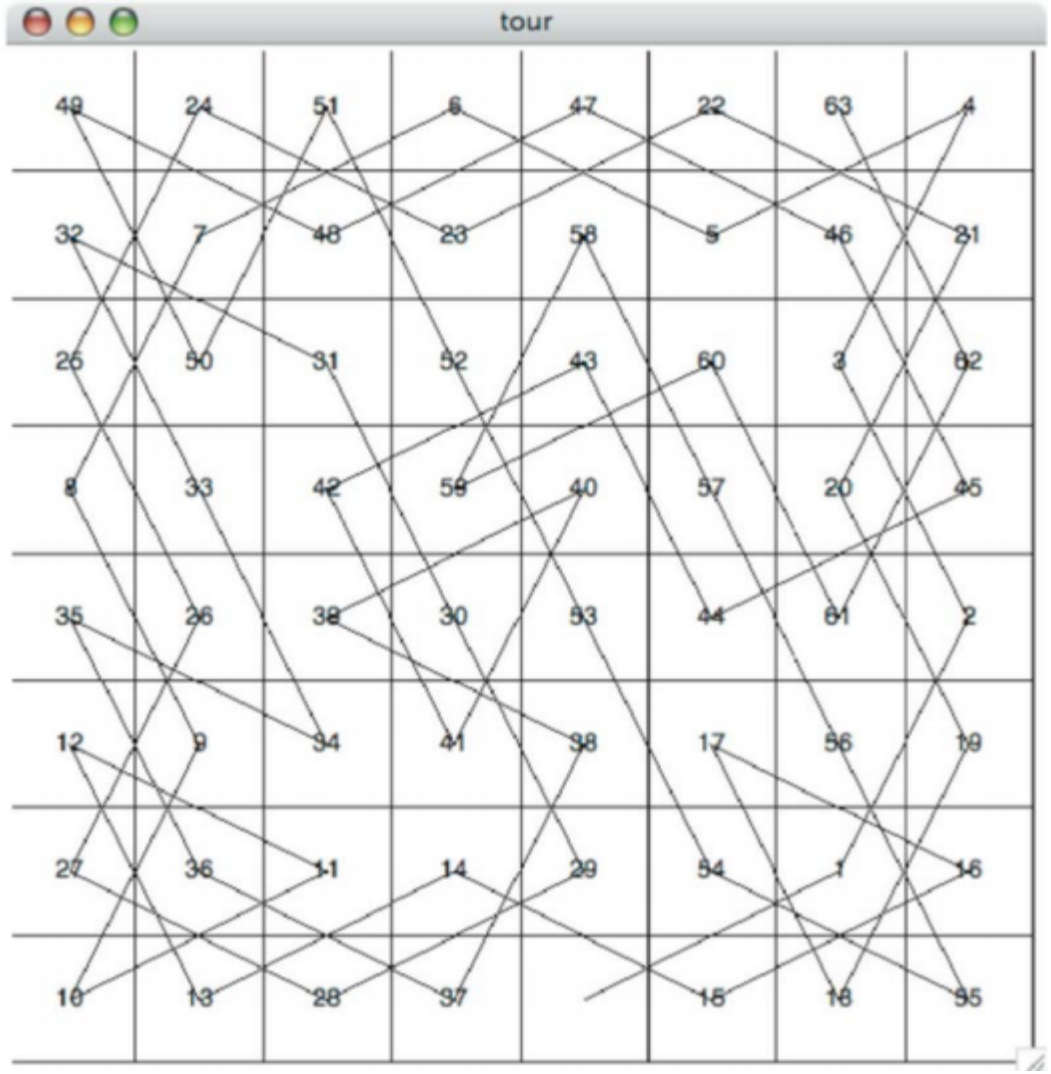
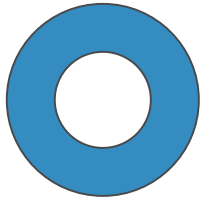


Figure 10: Finish

The Knight's Tour Problem: A Solution

骑士周游问题：一个解





Analysis and Improvement of the Algorithm for the Knight's Tour Problem

骑士周游问题算法分析与改进

Analysis of the Knight's Tour Algorithm
骑士周游算法分析

The performance of the above algorithm is highly dependent on the board size:

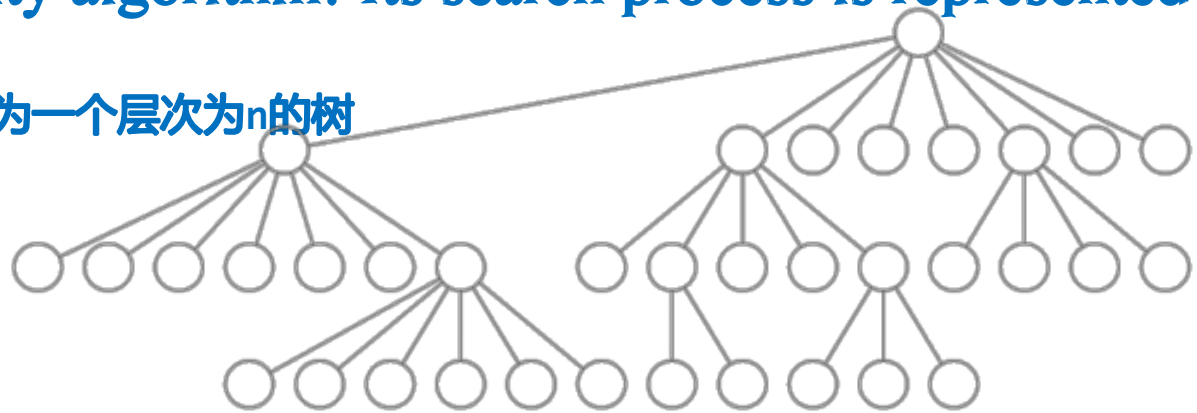
上述算法的性能高度依赖于棋盘大小：

- For a 5×5 chessboard, a travel path can be obtained in about 1.5 seconds
  就5×5棋盘，约1.5秒可以得到一个周游路径
- But for an 8×8 chessboard, it takes more than half an hour to get a solution
  但8×8棋盘，则要半个小时以上才能得到一个解

The algorithm currently implemented has a complexity of $O(k^n)$, where n is the number of checkerboards

目前实现的算法，其复杂度为$O(k^n)$，其中n是棋盘格数目

- This is an exponential time complexity algorithm! Its search process is represented as a tree with level n
  这是一个指数时间复杂度的算法！其搜索过程表现为一个层次为n的树



Improvement of knight tour algorithm
骑士周游算法改进

Fortunately, even exponential time complexity algorithms can be substantially improved in real-world performance

幸运的是，即便是指数时间复杂度算法也可以在实际性能上加以大幅度改进

- Smart construction of *nbrList* to arrange vertex access order in a specific way

  对nbrList的灵巧构造，以特定方式排列顶点访问次序

- It can reduce the search time of the travel path of the 8×8 chessboard to the second level!

  可以使得8×8棋盘的周游路径搜索时间降低到秒级！

This improved algorithm is specially named after the inventor: *Warnsdorff algorithm*

这个改进算法被特别以发明者名字命名：Warnsdorff法

Improvement of knight tour algorithm
骑士周游算法改进

In the initial algorithm, *nbrList* directly determines the branch order of depth-first search in the original order.

初始算法中nbrList，直接以原始顺序来确定深度优先搜索的分支次序

The new algorithm only modifies the order in which the next grid is traversed

新的算法，仅修改了遍历下一格的次序

Sort u's legal moving target checkerboard as: first search for the grid with the fewest legal moving targets

将u的合法移动目标棋盘格排序为：具有最少合法移动目标的格子优先搜索

```
def orderByAvail(n):
    resList = []
    for v in n.getConnections():
        if v.getColor() == 'white':
            c = 0
            for w in v.getConnections():
                if w.getColor() == 'white':
                    c = c + 1
            resList.append((c,v))
    resList.sort(key=lambda x: x[0])
    return [y[1] for y in resList]
```

Improvement of knight tour algorithm
骑士周游算法改进

The practice of using prior knowledge to improve algorithm performance is called "heuristic"

采用先验的知识来改进算法性能的做法，称作为“启发式规则”

Heuristic rules are often used in the field of artificial intelligence;

启发式规则经常用于人工智能领域；

It can effectively reduce the search range, reach the goal faster, etc.; for example, the chess program algorithm will pre-store "heuristic rules" such as chess manuals, formation formulas, master habits, etc., which can make moves from a large number of chess positions in the shortest time. Click to locate the best move in the search tree. For example: the "Golden Corner and Silver Edge" formula in Othello, the guidance program gives priority to the corner position, etc.

可以有效地减小搜索范围、更快达到目标等等；如棋类程序算法，会预先存入棋谱、布阵口诀、高手习惯等“启发式规则”，能够在最短时间内从海量的棋局落子点搜索树中定位最佳落子。例如：黑白棋中的“金角银边”口诀，指导程序优先占边角位置等等