DeepWalk: Online Learning of Social Representations

通过在网络中随机游走捕获网络局部信息，将游走序列等效为句子，并用Skip-Gram学习Embedding向量，从而完成网络表示学习

适应性： 由于网络是动态的，网络表示应适应网络的变化

社区意识*： 节点间的编码距离应能反应出成员之间的社会相似性

低维度： 维度低占用内存小，泛化能力强

连续性： 连续的表征不仅可以提供节点的可视化，也可以在分类时更加健壮性

随机游走

随机游走是个随机的过程，每次在网络中心行走时的选择都是随机的，无法基于过去的行为预测未来的行为。可以并行化，网络发生细微变化时只需要重新提取局部网络结构

语言模型

简单看下语言模型，给定一个单词序列：

$$W_1^n = (w_0, w_1, w_2, \ldots, w_n)$$

我们的目标是利用先前的单词去预测下一个单词：

$$p(w_n | w_0, w_1, w_2, \ldots, w_{n-1})$$

将其扩展到网络图中，以定长随机游走来探索网络结构，并根据先前走过的节点来预测下一个节点 v 的可能性：

$$p(v_i | (v_1, v_2, \ldots, v_{i-1}))$$

这里要注意，我们的目标是获取节点的 Embedding 向量，而不仅仅是求解节点共现概率，所以在这里引入映射矩阵 $\Phi : v \in V \to R^{|V| \times d}$ ，$\Phi(v)$ 代表节点 v 映射后的 d 维向量，有：

$$p(v_i | (\Phi(v_1), \Phi(v_2), \ldots, \Phi(v_{i-1})))$$

但随着随机游走的长度增加，概率函数没法计算。这时我们引入 Word2Vec 的 Skip-Gram 算法，利用一个节点来预测周围的节点，于是有：

$$minimiaze \ -log([v_{i-w}, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{i+w}] | \Phi(v_i))$$

注：这里的预测是无序性的，不考虑上下文相对给定单词的偏移量， Word2Vec 的顺序无关性也恰好契合随机游走的无序性。

通过将定长随机游走和 Word2Vec 相结合得到了一个满足所有要求的算法，该算法可以生成低维的网络表征，并存在于连续的向量空间中。

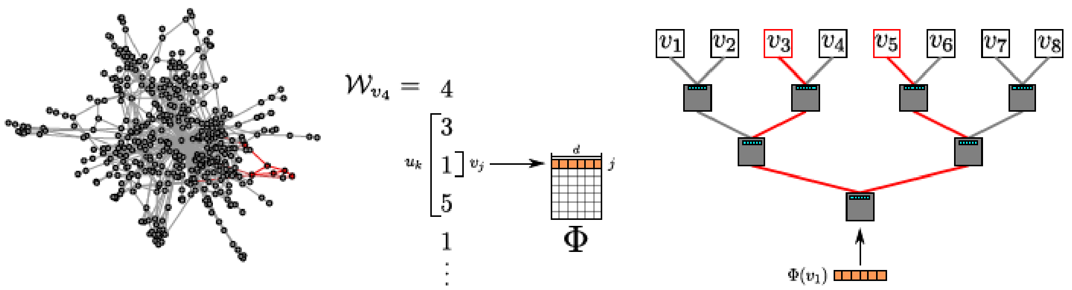
下图展示了我们的算法，3-9 行为核心算法：

- 外层循环打乱遍历顶点的顺序（每个节点都会作为随机游走的起点），这样相比于从固定顺序的顶点开始随机游走而言可以加速收敛；
- 内层循环中，我们会遍历序列中的所有的顶点，通过对每个节点进行均匀采样产生定长的随机游走序列，并通过 Skip-Gram 算法完成训练。

Algorithm 1 DeepWalk(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V|\times d}$

```
1: Initialization: Sample Φ from U^{|V|×d}
2: Build a binary Tree T from V
3: for i = 0 to γ do
4:     O = Shuffle(V)
5:     for each v_i ∈ O do
6:         W_{v_i} = RandomWalk(G, v_i, t)
7:         SkipGram(Φ, W_{v_i}, w)
8:     end for
9: end for
```



(a) Random walk generation. (b) Representation mapping. (c) Hierarchical Softmax.

Figure 3: Overview of DeepWalk. We slide a window of length $2w + 1$ over the random walk $\mathcal{W}_{v_4}$, mapping the central vertex v_1 to its representation $\Phi(v_1)$. Hierarchical Softmax factors out $\Pr(v_3 \mid \Phi(v_1))$ and $\Pr(v_5 \mid \Phi(v_1))$ over sequences of probability distributions corresponding to the paths starting at the root and ending at v_3 and v_5. The representation Φ is updated to maximize the probability of v_1 co-occurring with its context $\{v_3, v_5\}$.

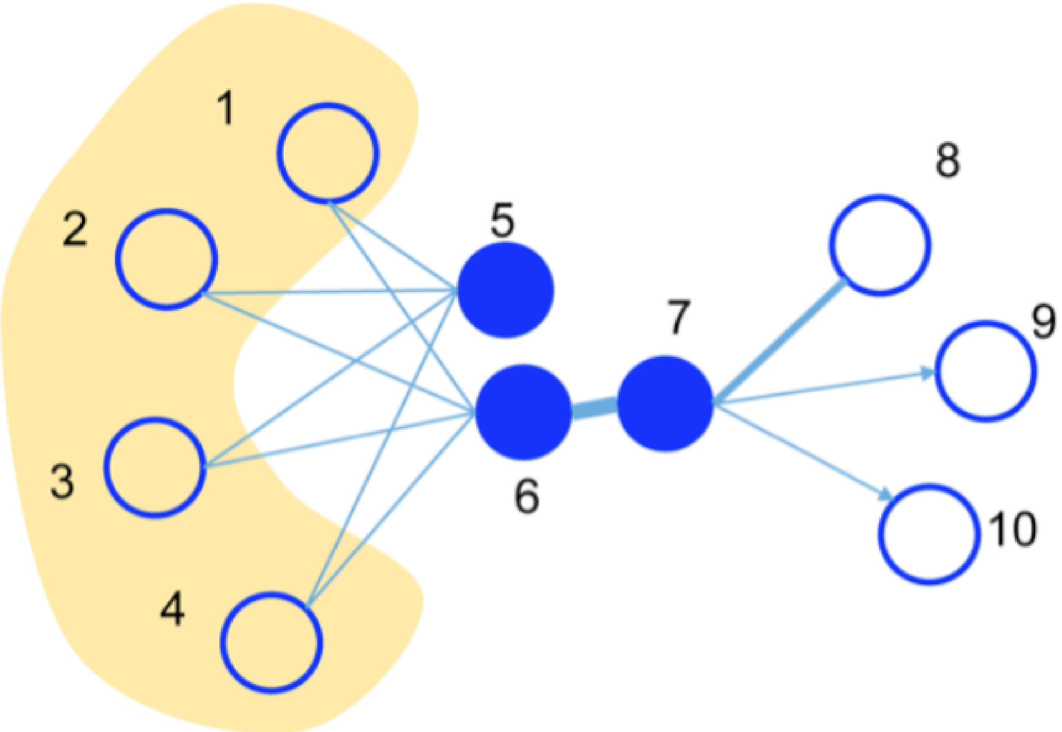
LINE:Large-scale Information Network Embedding

LINE模型致力于将这种大型的信息网络嵌入到低维的向量空间中，且该模型适用于任何类型(有向、无向亦或是有权重)的信息网络。并提出了一种解决经典随机梯度下降限制的边缘采样算法，提高了推理的有效性和效率。可用于可视化，节点分类以及关系预测等方面。

DeepWalk采用分布式并行方式来训练模型，但不适合大型网络

以前的网络没有捕捉到节点间更多的关系，只有一阶邻居相似性

LINE提出了**二阶相似性**，不是通过节点间的强弱来判定的，而是**通过节点的共享邻域结构来确定相似性。**



一方面，节点6和7之间的权值比较大，所以具有较高的一阶相似性，他们之间的嵌入向量距离比较近。另一方面，5,6虽然没有联系，但是他们有很多共同邻居，二阶相似性高，所以他们的嵌入向量也应该近。

一阶邻居

first-order 是指网络中节点之间的局部连接，我们对每条无向边进行建模并给出联合概率：

$$p_1(v_i, v_j) = \frac{1}{1 + exp(-u_i^T, u_j)}$$

其中，v_i 表示节点 i，u_i 为节点 i 对应的 Embedding 向量。

根据网络权值，我们的也有经验分布为：

$$\hat{p_1}(i, j) = \frac{w_{ij}}{W}$$

其中，w_{ij} 为节点 i 和结点 j 之间的权值，W 为网络总权值之和。

为了保证网络具有 first-order，我们只需要让经验分布和联合概率分布的越相近越好，衡量两个分布差异的指标为 KL 散度，忽略常数后可以得到代价函数：

$$minimize\ O_1 = -\sum_{(i,j)\in E} w_{ij} log(p_1(v_i, v_j))$$

这里，first-order 的目标函数只适用于无向图，不适用于有向图，所以我们这里用的是无向边。

二阶邻居

我们给出节点间共现的概率为：

$$p_2(v_j|v_i) = \frac{exp(u_j^{'T}, u_i)}{\sum_{k=1}^{|V|} exp(u_k^{'T}, u_i)}$$

其中，u_i 表示节点 i 的 Embedding 向量，u_i' 表示节点 i 为上下文时的 Embedding 向量，|V| 表示上下文节点的数量。

如上所述，second-order 是假设在上下文中具有相似分布的顶点彼此相似。

second-order 的经验分布为：

$$\hat{p}_2(v_j|v_i) = \frac{w_{ij}}{degree_i}$$

其中，w_{ij} 为边的权重，$degree_i$ 为节点 i 的度数，$degree_i = \sum_{j\in N(i)} w_i k$，$N(i)$ 为节点 i 的邻居。

为了保证 second-order，我们需要让条件概率分布 $p_2(\cdot|v_i)$ 与经验分布 $\hat{p}_2(\cdot|v_i)$ 相近。因此我们有：

$$minimize\ O_2 = -\sum_{i\in V} \lambda_i d(\hat{p}_2(\cdot|v_i),\ p_2(\cdot|v_j))$$

其中，$d(\cdot,\cdot)$ 表示两个分布的距离；λ_i 表示节点的重要性，可以通过类似 PageRank 算法得到。

接着用 KL 散度来代替 $d(\cdot,\cdot)$ 来衡量两个分布的相似性。为简单起见，我们令 $\lambda_i = degree_i$，所以我们有：

$$minimize\ O_2 = -\sum_{(i,j)\in E} w_{ij} log(p_2(v_j|v_i))$$

分别训练一阶近似和二阶近似的模型，然后将其得到的嵌入连接起来

负采样

目标函数为:

$$log\,\sigma(u_j^{'T}\cdot u_i) + \sum_{i=1}^{K} E_{ns}[log\,\sigma(-u_n^{'T}\cdot u_i)]$$

当然，涉及到稀疏参数更新，就可以利用异步随机梯度下降（Asynchronous Stochastic Gradient Algorithm, ASGD）算法进行加速。目标函数的偏导数为：

$$\frac{\partial O_2}{\partial u_i} = w_{ij}\cdot\frac{\partial\, p_2(v_j|v_i)}{\partial u_i}$$

可以看到计算梯度时需要乘上**边的权值**，但这样会出现一个问题：

- 如果选择一个较小的学习率，对于权值较小的边可能会导致梯度消失，学习速度过慢而无法收敛；
- 如果选择一个较大的学习率，对于权值较大的边可能会出现梯度爆炸。

所以，**该如何设定一个较好的学习率以应对边的权值方差较大的现象**？

边的权重与负采样

比较直接的想法是：导致这种问题的原因是边的权值，如果另所有边的权值相等就不会在出现这种问题了。

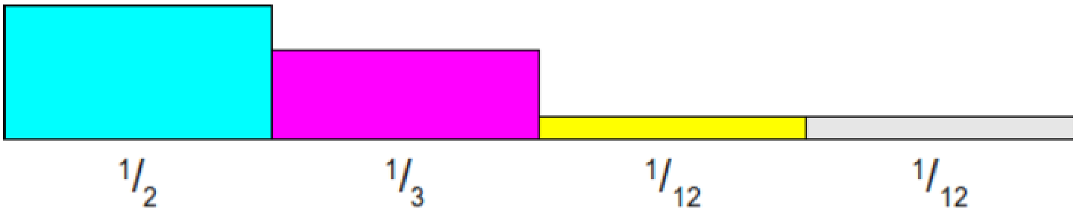
一个简单的方法就是将一个加权边分成多个权值为 1 的二元边，例如：一个权值为 4 的边，我们可以将其分成 4 个权值为 1 的二元边。

但这样又会出现新的问题：内存开销过大。为了解决这个新的问题，作者给出新的解决方案：**对原始边进行了采样，保证采样概率与原始边的权值成正比，并将采样后的边视为权值为 1 的二元边。**

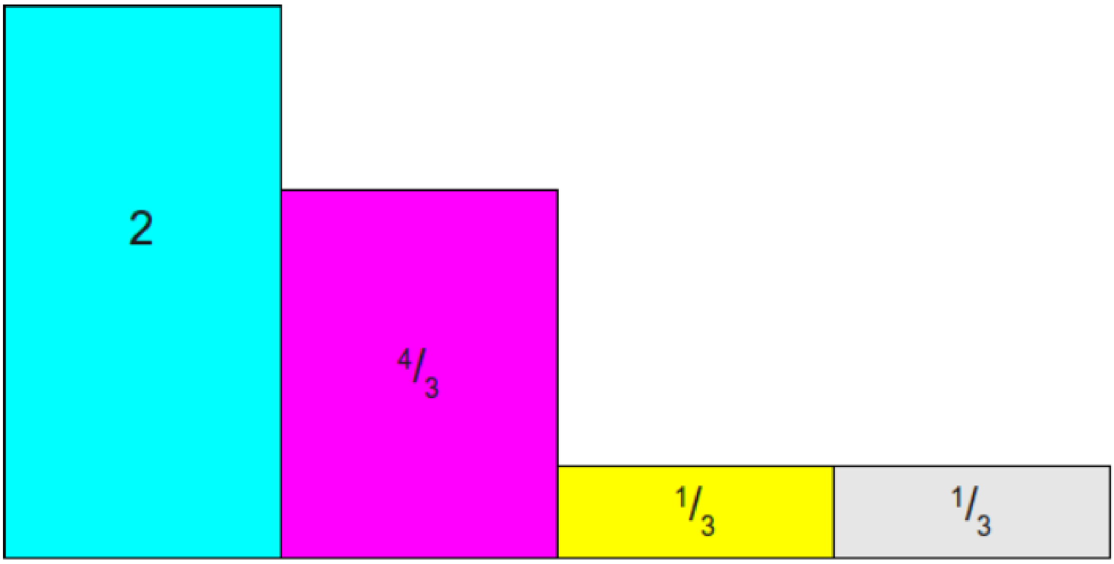
通过这种边采样处理，可以保证原本的代价函数不变，且又加入了边的权重信息。

关于加权采样问题，作者使用的 Alias 算法，虽然 Alias 非本文重点，但是我决定还是简单介绍一下。

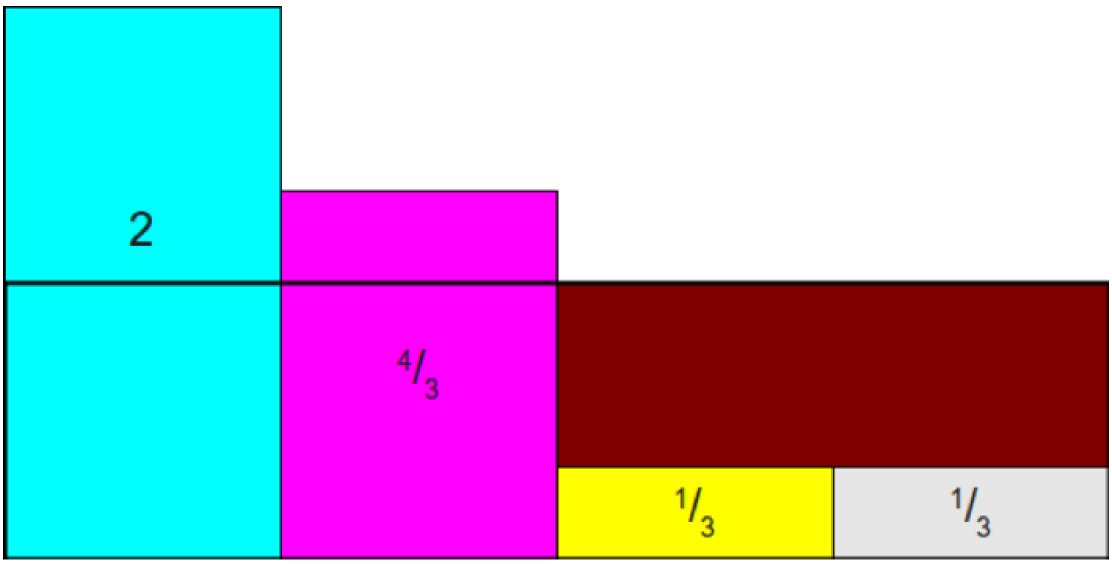
假设我们有四个权值：$\frac{1}{2},\frac{1}{3},\frac{1}{12},\frac{1}{12}$，现在要对其进行加权采样。



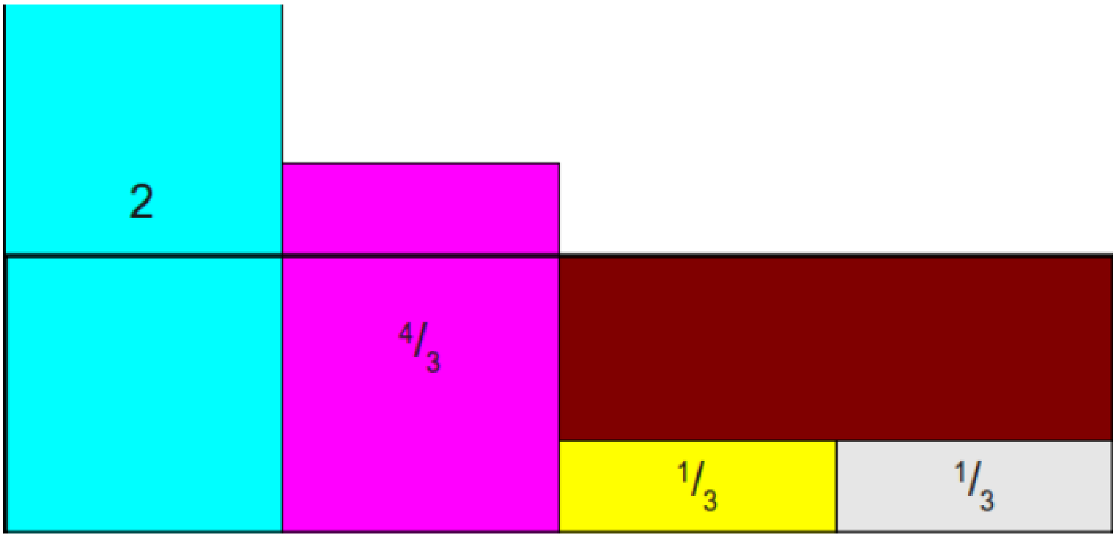
区别于利用最大值进行归一化，我们基于平均值进行归一化，而给出的例子的均值为$\frac{1}{4}$，所以有：



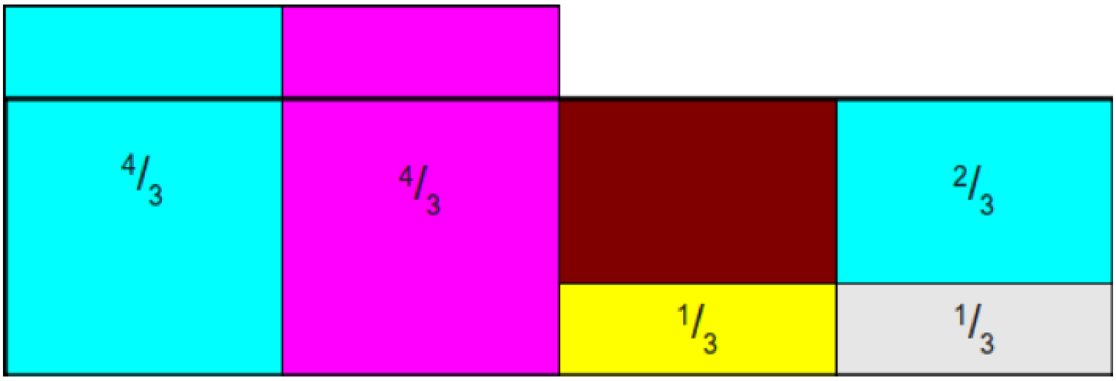
我们以均值归一化后的结果的新均值 1 为高度，画出一个矩形：



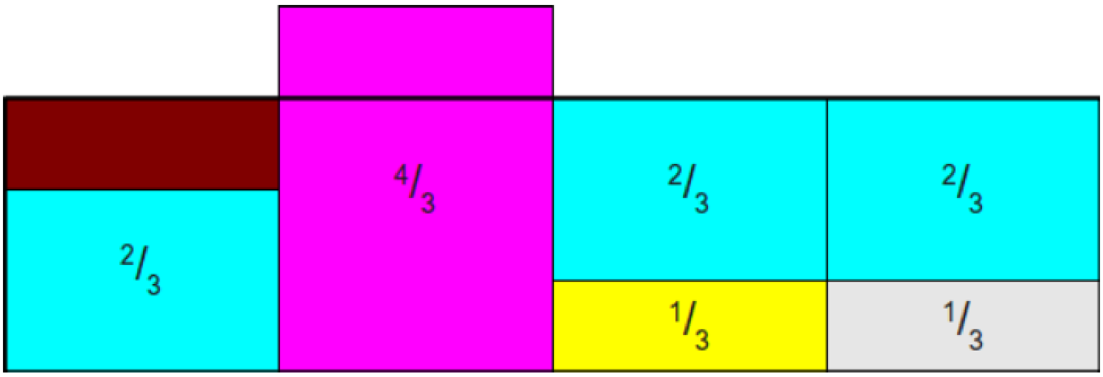
我们可以将多出的部分填补到空缺的部分：



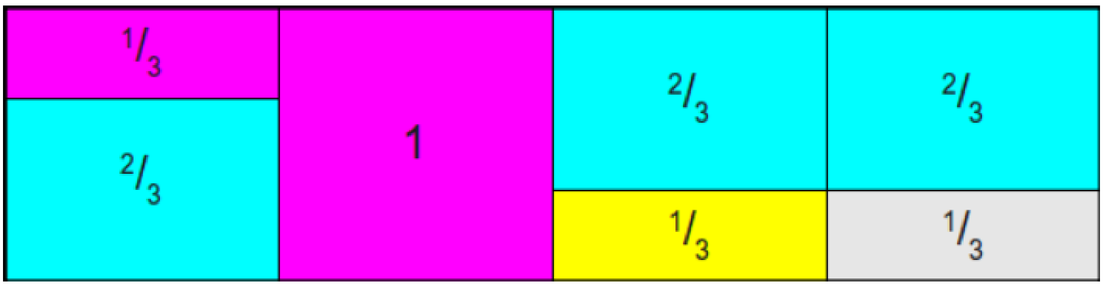
我们可以将多出的部分填补到空缺的部分：



现在还有两个多出来的部分，但只有一个空缺点。为了不增加开销，我们需要给出约束条件：**一列最多只有两个事件**，所以：



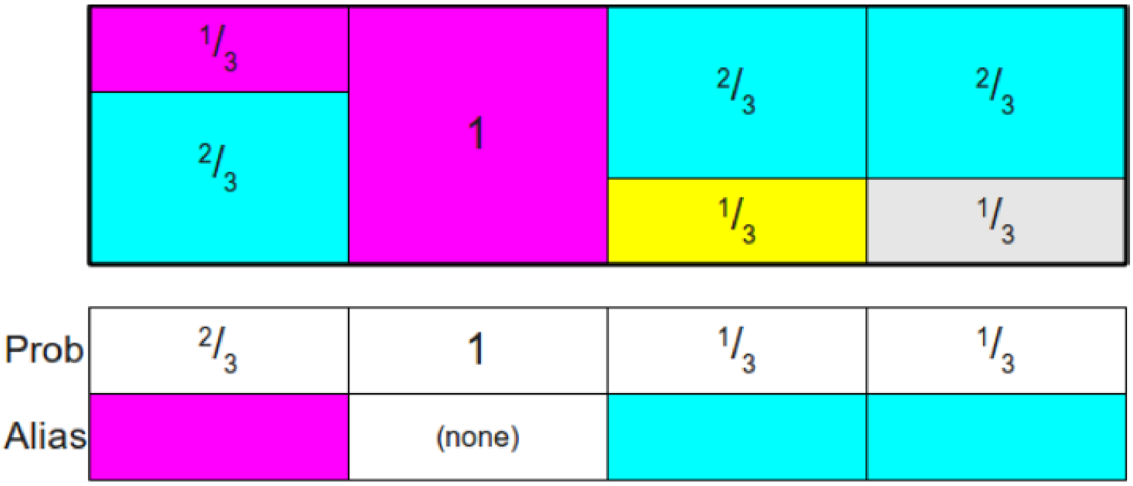
最后便产生了一个完整的矩阵：



我们来看下这个矩阵怎么使用。

我们构造两个大小相同的数组分别为概率表 Prob 和别名表 Alias，概率表为原始列在现有情况下的概率，如原先概率值为 $\frac{1}{2}$ 的第一列对应现在的概率值为 $\frac{2}{3}$，概率值为 $\frac{1}{3}$ 的第二列对应的现在的概率值为 1；而别名表 Alias 为多出来的另一个事件的概率，比如 Alias[0] 对应第二个事件， Alias[1] 为 None， Alias[2] 对应第一个事

件，Alias[3] 也对应第一个事件。



使用方法是，先随机到某一列，然后再进行一次随机用于判断是当前列的原本事件还是别名表 Alias 里面的另一个事件。比如我们第一次随机并得到第三列，有 Prob[2] = 1/3，然后再进行一次随机，如果随机数小于 1/3 则为事件三，如果随机数大于 1/3 则为 Alias[2] 中的别名事件，也就是事件一。

Node2vec: Scalable Feature Learning for Networks

采用有偏的随机游走算法并结合Skip-gram算法学习表示，通过超参数来设置搜索策略

同质性：属于同一集群的节点更加相似，如S1和U

结构等价性：两个具有相似结构的节点更加相似，如S6和U

struc2vec: Learning Node Representations from Structural Identity

定义了层次结构相似度,专注于节点结构性信息进行Embedding,在每层上用有偏的随机游走获得节点序列,用Word2Vec训练Embedding

关注**不同的节点在网络中的所处的角色**

DeepWalk或node2vec这一类的方法在判断节**点的结构是否等价**的分类任务上往往并不能取得好的效果。其根本原因在于网络中的节点具有**同质性**（homohily），即两个节点有边相连是因为它们有着某种十分相似的特征。因此在网络中相距比较近的节点在嵌入空间也比较近，因为他们有着共同的特征；而在网络中相距比较远的节点，则认为它们没有共同特征，因此在嵌入空间的距离也会比较远，尽管两个节点可能在局部的拓扑结构上是相似的。

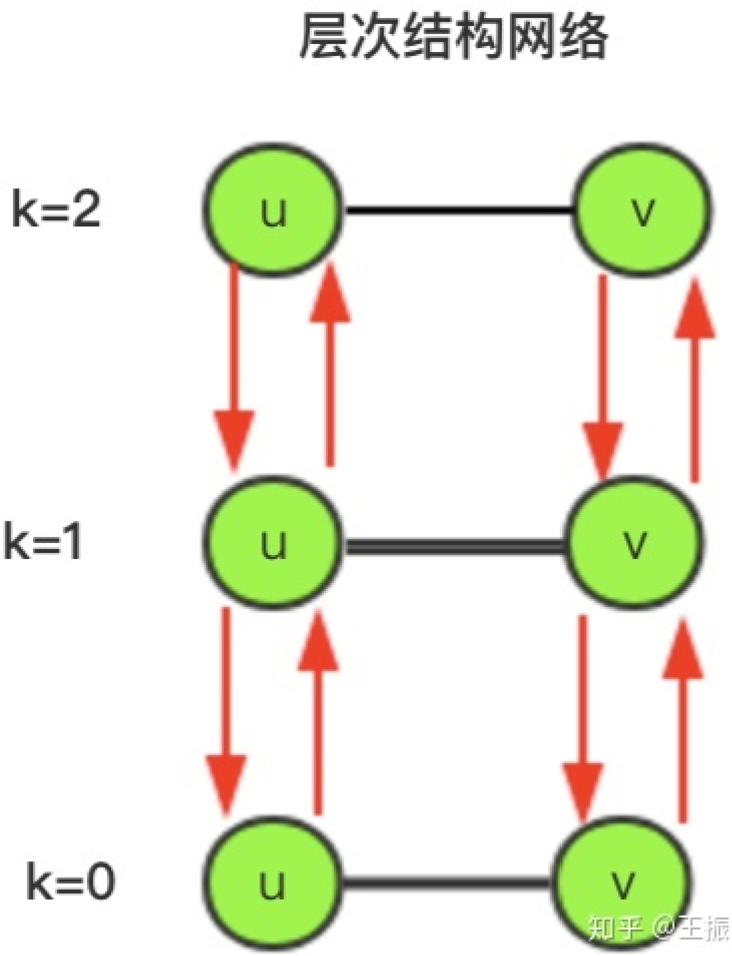
一个好的可以反映节点结构特性的方法必须满足以下两个特征：

- 嵌入空间的距离得能反映出节点之间的结构相似性，两个局部拓扑结构相似的节点在嵌入空间的距离应该相近。
- 节点的结构相似性不依赖于节点或边的属性甚至是节点的标签信息。

算法可以分成四步：

1. 根据不同距离的邻居信息分别算出每个节点对的结构相似度，这涉及到了不同层次的结构相似度的计算。

2. 构建一个多层次的带权重网络M，每个层次中的节点皆由原网络中的节点构成。
3. 在M中生成随机游走，为每个节点采样出上下文。
4. 使用word2vec的方法对采样出的随机游走序列学习出每个节点的节点表示。



- 每一层节点相同,一共k*V个节点
- 包含两种边的类型,**黑色**为第k层节点u与v的层次结构相似性,**橙色**为层间的权重

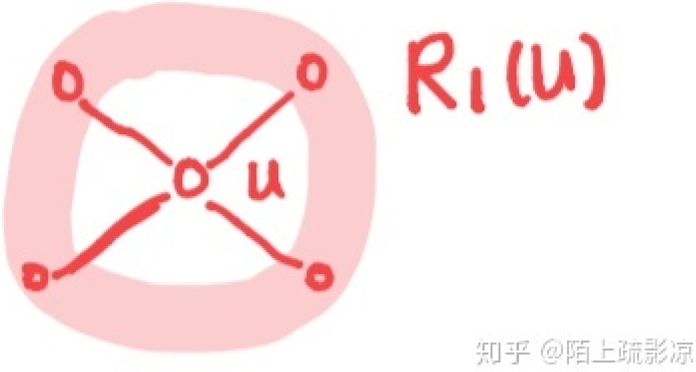
符号

$G=(V,E)$：无向带权网络，V表示节点集合，E表示边集合。

$n=|V|$：网络中的节点数。

k^*：网络的直径，即网络中任意两点距离的最大值。

$R_k(u)$：与节点u距离为k的节点集合，等同与以u为根的BFS树上第k层的节点集合。例如，$R_1(u)$就是u的直接邻居。



$s(S)$：对某个节点集合V中的节点按照度的从小到大顺序排序后形成的序列。

$f_k(u,v)$：考虑两个节点的k跳邻域（k-hop neighborhoods）时（小于等于k跳的所有邻居均要考虑），两个节点的在结构距离（structural distance）。表达式如下：

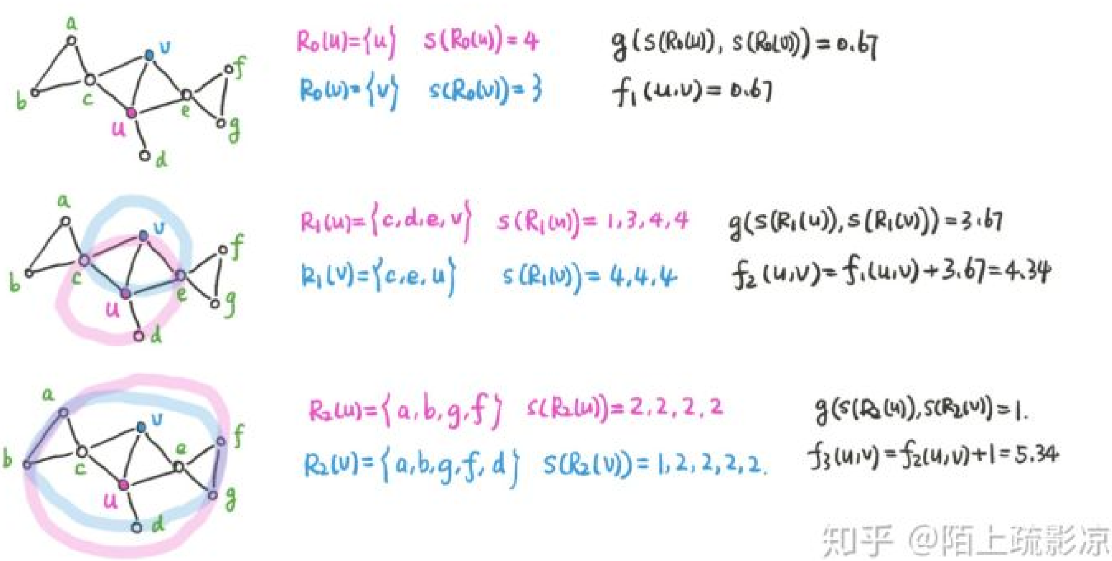
$$f_k(u,v) = f_{k-1}(u,v) + g\left(s\left(R_k(u)\right), s\left(R_k(v)\right)\right)$$
$$k \geq 0 \text{ and } \left|R_k(u)\right|, \left|R_k(v)\right| > 0$$

$f_k(u,v)$ 就是距离u，v相距为k的那些节点之间的结构距离。这是一个递归定义，$f_{k-1}(u,v)$ 表示考虑k-1跳邻域时的距离，再加上只考虑k跳邻居的距离，就形成了k跳邻域的距离了，初始值 $f_{-1} = 0$。

$g(D_1, D_2)$ 表示两个有序的序列 D_1, D_2 的距离。$s(R_k(u)), s(R_k(v))$ 分别表示与u，v距离为k的节点按照度大小排序后的度序列。

注意到$f_k(u,v)$的计算是在$f_{k-1}(u,v)$上加上一个非负的值，因此**该函数关于k是一个单调不降的函数**。并且这个函数只有在两个节点同时存在k跳邻域的时候才有定义。

下面给出一个具体的例子：



设计思想

两节点度相同,结构就相似,若邻居还有相同的度,就更相似

节点u,v之间的距离

节点u,v在前k层的距离$f_k(u,v)$前k-1层的距离 $+$ k阶邻居的度序列的距离

g:节点u,v的k阶邻居节点的度序列相似度

$$f_k(u,v) = f_{k-1}(u,v) + g(s(R_k(u)), s(R_k(v))),$$
$$k \geq 0 \text{ and } |R_k(u)|, |R_k(v)| > 0$$

R_k():节点的k阶邻居节点的度的列表; s()表示序列化

- g()是DTW算法,计算两个不一致序列的相似度

相似度

距离越小,w越大

$$w_k(u,v) = e^{-f_k(u,v)}, \quad k = 0, \ldots, k^*$$

指数权重归一化，距离越大，越不相似

归一化

$$p_k(u,v) = \frac{e^{-f_k(u,v)}}{Z_k(u)}$$

不同层之间的边权重

若u在当前层有很多相似的节点,则应该更往上一层,获得更细的信息去区分,所以k$k- > k+1$的权重更大

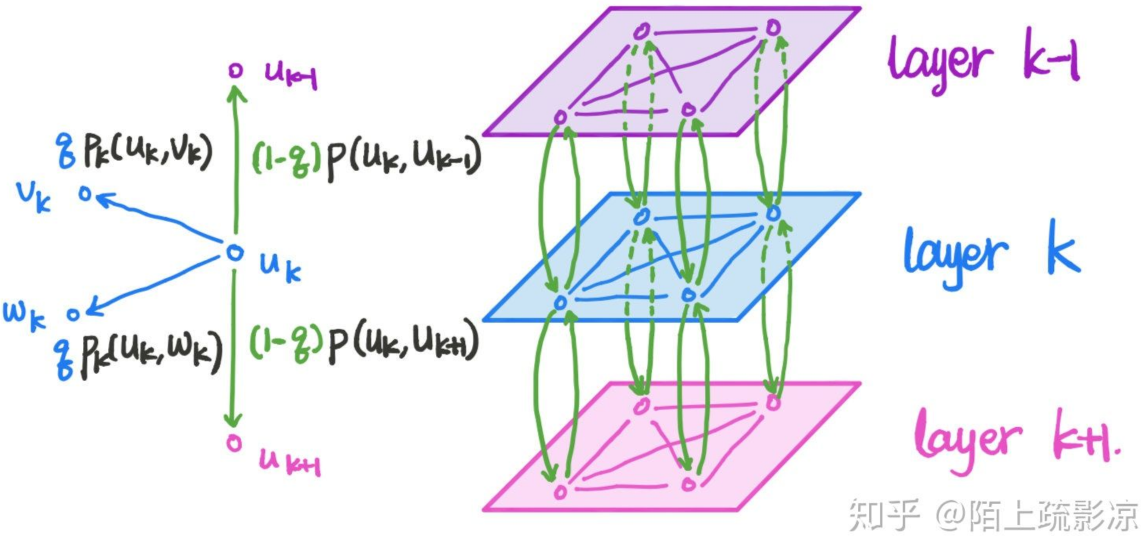
$$w(u_k, u_{k+1}) = \log(\Gamma_k(u) + e), \quad k = 0, \ldots, k^* - 1$$
$$w(u_k, u_{k-1}) = 1, \quad k = 1, \ldots, k^*$$

- 如果u在当前层有很多相似的节点,可能当前信息区分度不够，应该更往上一层，获得更多的信息去区分。
- $\Gamma_k(u) = \sum_{v \in V} I(w_k(u,v) > \bar{w_k})$表示节点u在第k层具有相似节点的数量,相似节点的定义为权重大于第k层平均边权的邻居

其中$\Gamma_k(u)$表示第k层中，所有指向u的边中权重大于该层平均权重的数量。具体的式子:

$$\Gamma_k(u) = \sum_{v \in V} 1\left(w_k(u,v) > \overline{w_k}\right)$$

$\bar{w}_k$ 第k层所有边权的平均值。$\Gamma_k(u)$ 实际上表示了第k层中，有多少节点是与节点u相似的，如果u与很多节点都相似，说明此时一定处于低层次，考虑的信息太少，那么$\Gamma_k(u)$将会很大，即$w(u_k, u_{k+1}) > w(u_k, u_{k-1})$，对于这种情况，就不太适合将本层中的节点作为上下文了，应该考虑跳到更高层去找合适的上下文，所以去喜高层的权重更大。



归一化

$$p_k(u_k, u_{k+1}) = \frac{w(u_k, u_{k+1})}{w(u_k, u_{k+1}) + w(u_k, u_{k-1})}$$

$$p_k(u_k, u_{k-1}) = 1 - p_k(u_k, u_{k+1})$$

随机游走

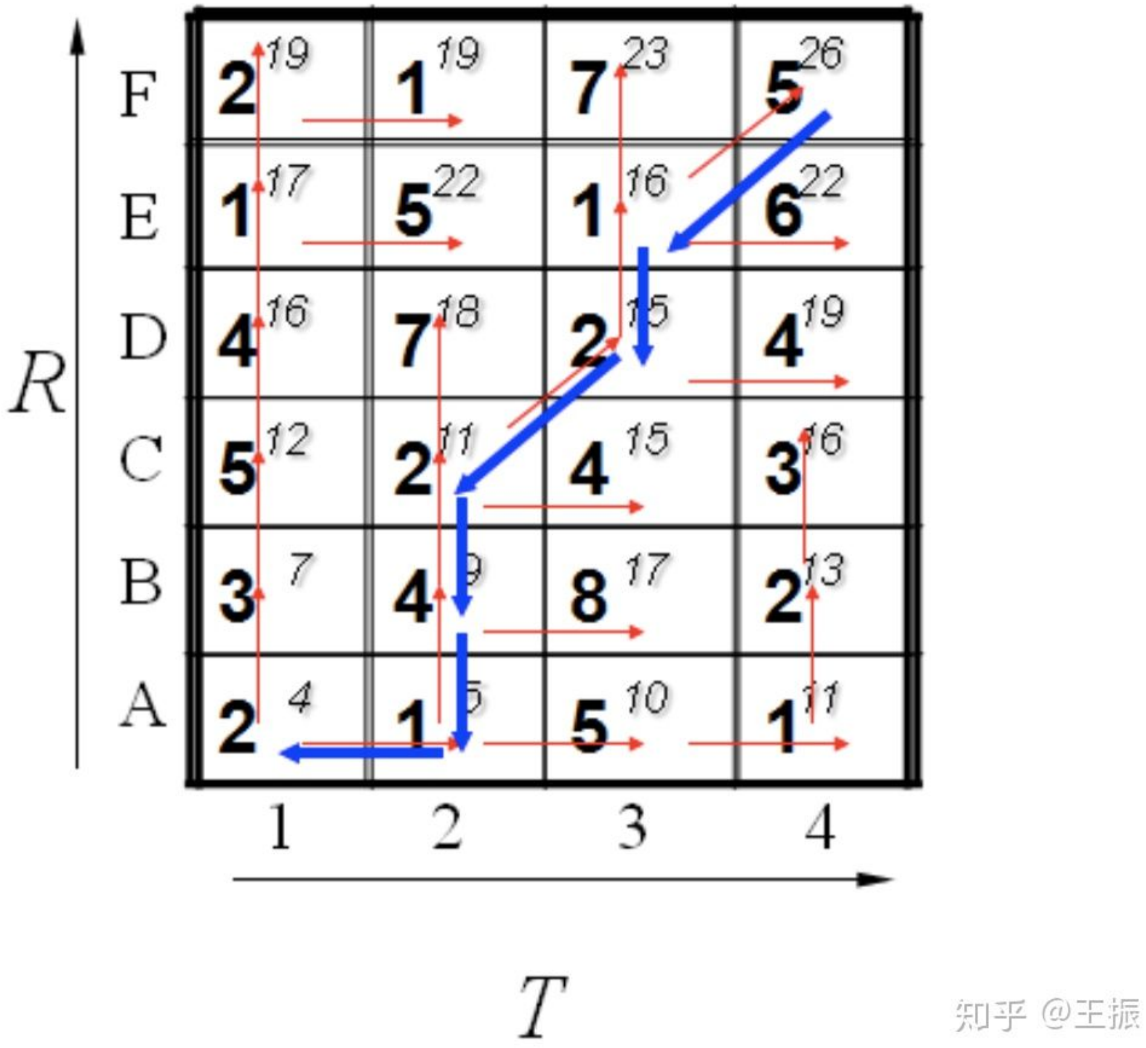
设置随机游走步数、随走游走器个数、当前层游走的概率，对每个随机游走器：

1. 从第0层开始，先某一概率q决定是在当前层游走 还是改变层。
2. 若在当前层游走，则通过权重 $P_k(u, v)$ 有偏游走。
3. 若改变层，通过中 $P_k(u_k, u_{k+1}), P_k(u_k, u_{k-1})$ 决。定是往上层还是往下层
4. 将每一层上访问到的节点加到上下文中（不考虑所处层）。
5. 直至满足停止条件：达到设定的游走步数。

DTW

DTW语音处理中衡量两个长度不一致的有序序列相似度比较经典的方法，是一种动态规划的方法，规整两个序列，使长度保持一致，再计算相似度。

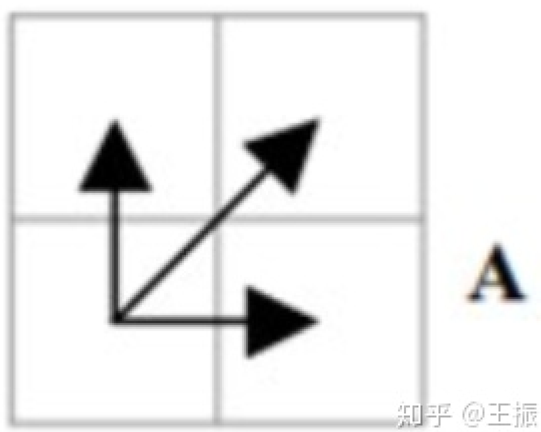
算法思路：是将两个序列组成矩阵网格，寻找一条通过该网格中若干个点的路径，路径通过的格点即为两个序列进行计算的对齐的点。 这个条路径长度可以作为即为两个序列的相似度衡量



- X、Y轴分别为长度不一致的u、v节点k阶邻居度序列index
- 单元格大数字d(i,j): 表示两个序列index对应元素的距离 / 相似度公式：

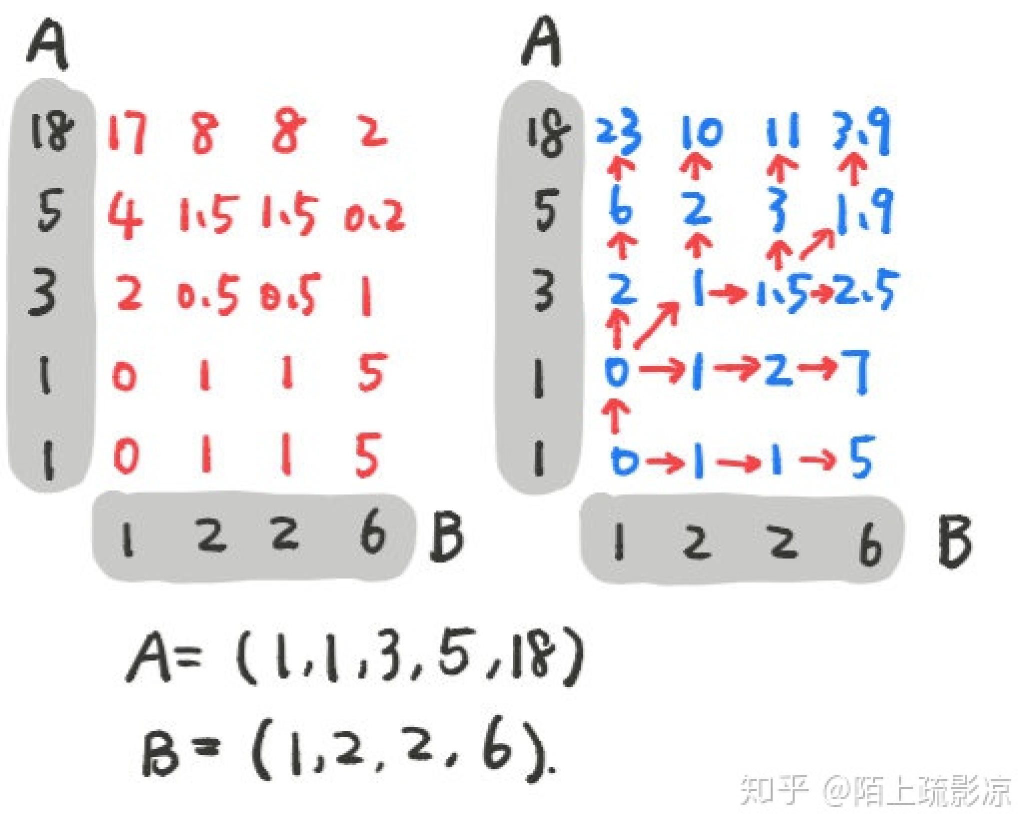
$$d(a,b)=\frac{\max(a,b)}{\min(a,b)}-1$$

-
  - 相比L1,L2距离，用该距离公式的优点是 认为：度为100与101距离，比度各自为1与2的距离是不同的，差异更小
- 单元格小数字g(i,j)表示：每一格，若从左下角出发，最少要路径可以达到。
-
  - 只能朝下图所示三个方向游走：



-
  - 到g(i,j)的最短路径长度计算

对于两个序列A、B，对任意的$a \in A, b \in B$定义一个距离函数$d(a,b)$表示a与b的距离，DTW想利用这样定义的距离函数找到序列A、B的最小距离。举个例子：



> 设A=(1,1,3,5,8),B=(1,2,2,6)是两个已经排序过的度序列，给出距离函数的定义为$d(a,b) = \frac{max(a,b)}{min(a.b)} - 1$，那么对A、B中所有元素两两之间计算距离d，得到左图中红色的距离。接下来我们利用这个矩阵来计算序列A、B的距离。
> 计算用到的算法是动态规划，递推式为：
> $$g(i,j) = \min \begin{cases} g(i-1,j) + d(i,j) \\ g(i-1,j-1) + 2d(i,j) \\ g(i,j-1) + d(i,j) \end{cases}$$
> 比如要计算$g(3,2)$,那么首先找到
> $g(2,2) = 1, g(2,1) = 0, g(3,1) = 2, d(3,2) = 0.5$,
> $g(2,2) + d(3,2) = 1 + 0.5 = 1.5$
> $g(2,1) + 2d(3,2) = 0 + 2 \times 0.5 = 1$
> $g(3,1) + d(3,2) = 2 + 0.5 = 2.5$
> 因此$g(3,2) = min1.5, 1, 2.5 = 1$，然后我们用一个箭头标注$g(3,2)$是从$g(2,1)$计算出来的。
> 通过这样逐个计算，直到下标到达A、B的最大长度，则$g(5,4)$就是A、B这两个序列的距离。
> 观察到这样一个现象，当A=(1,1,3,5)时，其实与B的度序列非常相似，此时计算出来两个序列的距离为1.9（上图的$g(4,4)$），而给A加了一项很大的值后，两个度序列的距离一下子就变成了3.9。

> 最后，为什么这么定义距离函数？看个例子，如果a=1，b=2，两者差值为1，$d(a,b)=1$；而若a=100，b=101，两者差值仍然为1，此时$d(a,b)=0.01$。这说明后者比前者距离更小，更相似。