4.最短路：标签法

4.1 介绍

最短路问题可以视为在无容量限制的网络中，从节点 s 向 N-{s}中的每个节点尽可能便宜地发送 1 个单位的流。

$$\text{Minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{4.1a}$$

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} n-1 & \text{for } i = s \\ -1 & \text{for all } i \in N - \{s\} \end{cases} \tag{4.1b}$$

$$x_{ij} \geq 0 \qquad \text{for all } (i,j) \in A. \tag{4.1c}$$

在研究最短路问题时，首先建立几个基本假设：

假设 4.1。所有弧长都是整数。

假设 4.2。该网络包含从节点 s 到网络中每个其他节点的有向路径。

假设 4.3。网络不包含负循环（即负长度的有向循环）。

假设 4.4。网络是定向的。

常见的最短路问题包括：

1。当弧长为非负时，寻找从一个节点到所有其他节点的最短路径。

2。求任意弧长网络中从一个节点到所有其他节点的最短路径

3。寻找从每个节点到其他节点的最短路径

4。最短路径问题的各种推广

. **标签设置算法和标签校正算法**：标签设置算法在每次迭代中将一个标签指定为永久标签。标签校正算法认为所有标签都是临时的，直到最后一步，它们都成为永久的。标签设置算法只适用于任意弧长的无环网络或弧长非负的最短路径问题，效率更高，即具有更好的最坏情况复杂度界限，可以把标签设置算法看作是标签校正算法的特例。

4.2 应用（什么样的问题能看做最短路问题）

1 近似分段线性函数

问题描述：用一个使用较少断点的近似函数替换分段线性函数能够节省存储空间和使用函数的成本；然而，由于近似函数的不精确性，我们将付出代价

设 f1（x）是变量 x 的分段线性函数。它通过 n 个点 a1=（X1，Y1），a2=（X2，Y2），an=（Xn，Yn），X1<=X2<=…<= Xn。函数在每两个连续点 Xi 和 Xi+1 之间线性变化。用另一个函数 f2（x）来近似函数 f1（x）

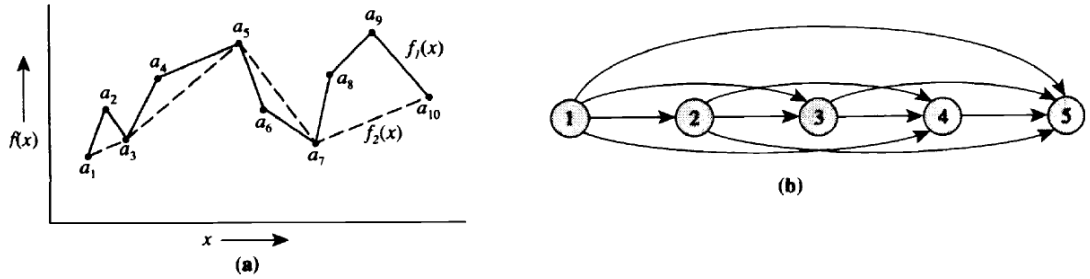


Figure 4.1 Illustrating Applications 4.1: (a) approximating the function $f_1(x)$ passing through 10 points by the function $f_2(x)$; (b) corresponding shortest path problem.

假设 ai 与 aj 之间的成本为α，近似误差与实际数据点和估计点之间的平方误差之和成正比（即，惩罚为 $\beta \sum_{i=1}^{n} [f_1(x_i) - f_2(x_i)]^2$，β 为某个常数）

因此，决策问题是确定用来定义近似函数 f2（X）的点的子集

可以把这个问题描述为一个网络 G 上的最短路径问题，网络 G 有 n 个节点，编号为 1 到 n。弧（i，j）的成本 cij 有两个组成部分：存储代价 α 和通过连接 ai 和 aj 的线上的对应点来近似 aj 和 aj 之间的所有点的代价。在区间[Xi，Xj]中，逼近函数是 f2（x）= f1（Xi）+（X-Xi）[f1（Xj）-f1（Xi）])/(Xj-Xi)，所以这个时间间隔内的总成本是

$$c_{ij} = \alpha + \beta\left[\sum_{k=i}^{j} (f_1(x_k) - f_2(x_k))^2\right].$$

G 中从节点 1 到节点 n 的每条有向路径对应于一个函数 f2（X），并且该路径的成本等于存储该函数以及使用它来近似原始函数的总代价。

2 在生产线上进行检验工作

一条生产线由 n 个生产阶段的有序序列组成，每个阶段都有一个生产操作和一个可能的检查操作，每个阶段都可能产生缺陷，在第 i 阶段产生缺陷的概率是 ai，所有的缺陷都是不可修复的。生产线的末端必须有一个检验站，这样就不会装运任何有缺陷的产品。

决策问题是找到一个最优的检验计划，规定应该在哪个阶段检验产品，以最小化生产和检验的总成本。

考虑以下成本数据：（1）Pi，第 i 阶段单位制造成本(2） fij，上一次检验是在第 i 阶段后，第 j 阶段后检验的固定成本；（3）gij，上一次检验是在第 i 阶段后，在阶段 j 之后检验一个项目的单位可变成本。j 站的检验成本取决于上一次检验批次的时间。

我们可以把这个检验问题描述为（n+1）个节点的网络上的最短路径问题，编号为 0，1，n。网络中从节点 0 到节点 n 的每条路径都定义了一个检验计划。例如，路径 0-i-n 意味着我们在第 i 和第 n 阶段之后检查批。假设 $B(i) = B\prod_{k=1}^{i}(1 - \alpha_k)$ 表示第 i 阶段结束时的预期无缺陷单元数

$$c_{ij} = f_{ij} + B(i)g_{ij} + B(i)\sum_{k=i+1}^{j} p_k. \qquad (4.2)$$

3 背包问题

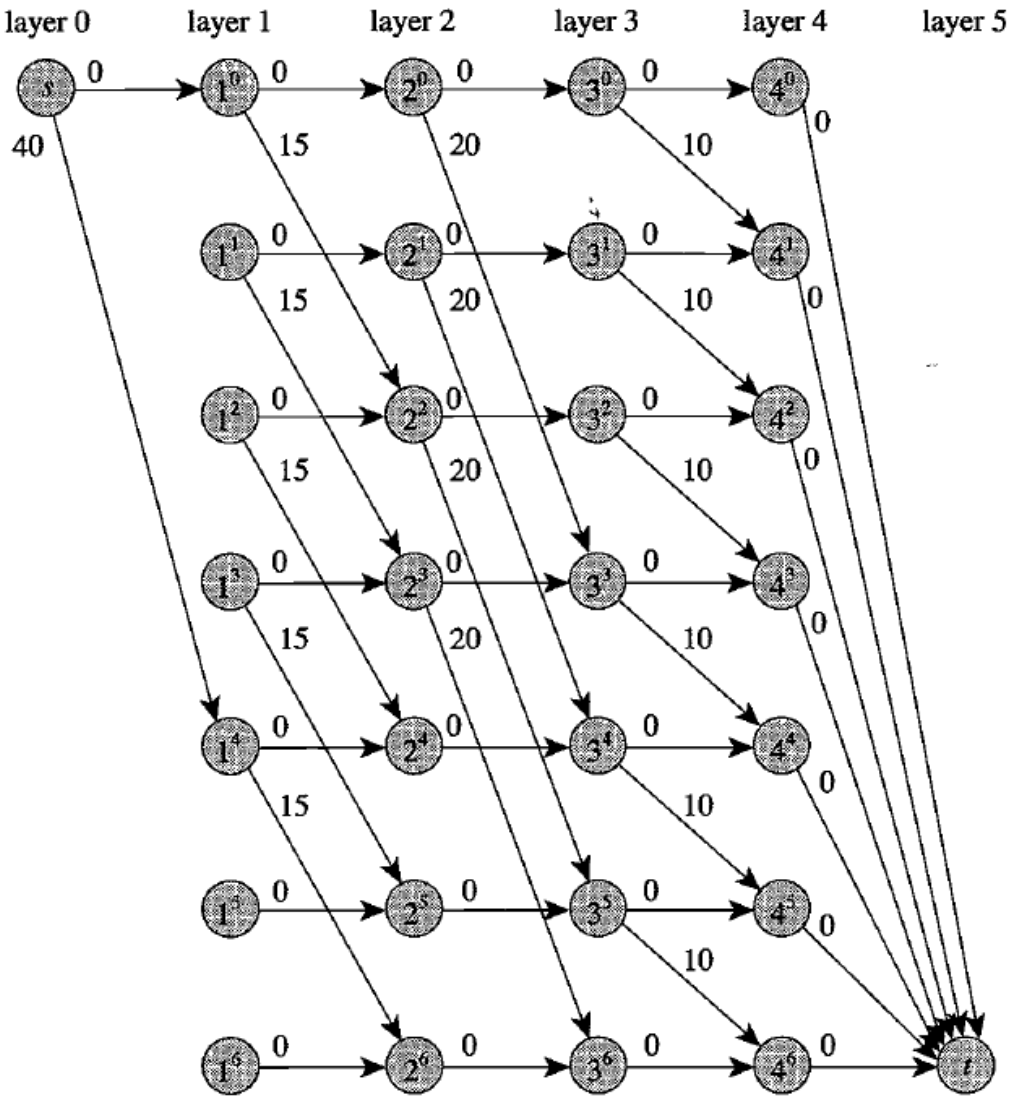


Figure 4.3 Longest path formulation of the knapsack problem.

假设背包的容量为 W=6。公式中的网络有 6 层节点：中间层对应于每个项目，一层对应于源节点 s，另一层对应于汇节点 t。对应于项目 i 的层有 W+1 个节点，i0，i1，…，iW。节点网络中的 ik 表示已经消耗了背包容量的 k 个单位。节点 ik 最多有两个传出弧，对应于两个决定：(1) 装入 (i+1)，2) 不装 i+1[只有当背包有足够的备用容量容纳物品 (i+1) 时，才能选择第二个方案]第一个决策对应的弧是效用为零的 ((ik，(i+1) k)，第二个决策对应的弧是 (ik，(i+1) k+w_{i+1}) 带有效益 U_{i+1}。源节点有两个关联弧 (s，1_0) 和 (s，1_{w1})，对应于是否将项目 1 包括在空背包中的选择。最后，我们将最后一项对应的层中的所有节点以零效用弧连接到节点 t。

背包问题的每一个可行解都定义了一条从节点 s 到节点 t 的有向路径

4 不定期船问题

一艘不定期轮船载着货物和乘客从一个港口开往另一个港口。轮船从 i 港到 j 港的航程赚取 Pij 单位的利润，需要 τij 单位的时间。当我们用下列表达式定义任何航程 W 的日利润时，船长想知道轮船的哪个航程 W（即一个有向循环）达到最大可能的平均日利润

$$\mu(W) = \frac{\sum_{(i,j)\in W} p_{ij}}{\sum_{(i,j)\in W} \tau_{ij}}.$$

5 差分约束系统

约束形式为 Ax<=b，n*m 约束矩阵 A 每行包含一个+1 和一个-1，所有其他值都为零。

$$x(j_k) - x(i_k) \leq b(k) \qquad \text{for each } k = 1, \ldots, m. \tag{4.3}$$

如，

$$x(3) - x(4) \leq 5, \tag{4.4a}$$
$$x(4) - x(1) \leq -10, \tag{4.4b}$$
$$x(1) - x(3) \leq 8, \tag{4.4c}$$
$$x(2) - x(1) \leq -11, \tag{4.4d}$$
$$x(3) - x(2) \leq 2. \tag{4.4e}$$

可以对其进行如下转换

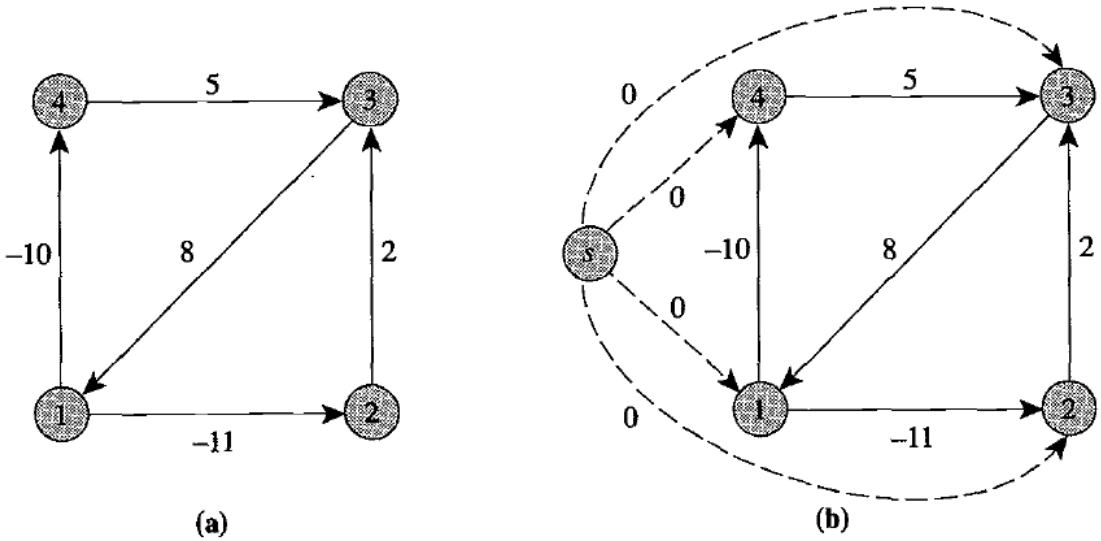


Figure 4.4 Graph corresponding to a system of difference constraints.

由于网络中存在负循环，不能用标签设置算法求解，可以使用标签校正算法来检测网络中是否存在负循环。标签校正算法要求所有节点都可以通过某个节点的有向路径到达，将该节点用作最短路径问题的源节点。为了满足这一要求，引入了一个新的节点，并以零成本弧连接到网络中的所有节点。

4.3 最短路径树

存储从源节点到所有其他（n-1）节点的最短路径并不需要（n-1）^{2}的存储空间，只需要（n-1）的存储空间，因为可以构造一个最短路径树，从源节点到其他节点的最短路径都包含在最短路径树中。

性质 4.1。如果路径 s=i1-i2-⋯-ih=k 是从节点 s 到节点 k 的最短路径，那么对于每个 q=2,3，h-1，子路径 s=il-i2-iq 是从源节点到节点 iq 的最短路径（好理解，证明略）

性质 4.2 当且仅当 d（j）=d（i）+cij，（i，j）∈p 时，从源节点到节点 k 的有向路径 P 是最短路径，

4.4 非循环网络中的最短路径问题

标签设置算法的最坏情况复杂度为 O（m），在非循环网络上求解最短路径问题的任何其他算法都不能再快了（就最坏情况的复杂性而言），因为求解问题的任何算法都必须检查每个弧，这本身就需要花费 O（m）时间。

到达算法

我们首先将 d（s）=0 和剩余的距离标签设置为一个非常大的数字。然后我们按照拓扑顺序检查节点，对于每个被检查的节点 i，我们扫描 A（i）中的弧。如果对于任意弧（i，j）∈a（i），d（j）>d（i）+Cij，令 d（j）=d（i）+Cij。当算法按此顺序检查所有节点一次时，距离标签是最优的。

定理 4.3。到达算法在 O（m）时间内解决了非循环网络上的最短路径问题。

4.5 DIJKSTRA 算法

djistra 算法的核心在于标签设定，分为临时标签和永久标签，每一步将一个临时标签转化为永久标签，当所有标签均为永久标签时算法结束。

```
algorithm Dijkstra;
begin
    S := ∅; S̄ := N;
    d(i) := ∞ for each node i ∈ N;
    d(s) := 0 and pred(s) := 0;
    while |S| < n do
    begin
        let i ∈ S̄ be a node for which d(i) = min{d(j) : j ∈ S̄};
        S := S ∪ {i};
        S̄ := S̄ − {i};
        for each (i, j) ∈ A(i) do
            if d(j) > d(i) + cij then d(j) := d(i) + cij and pred(j) := i;
    end;
end;
```

Figure 4.6 Dijkstra's algorithm.

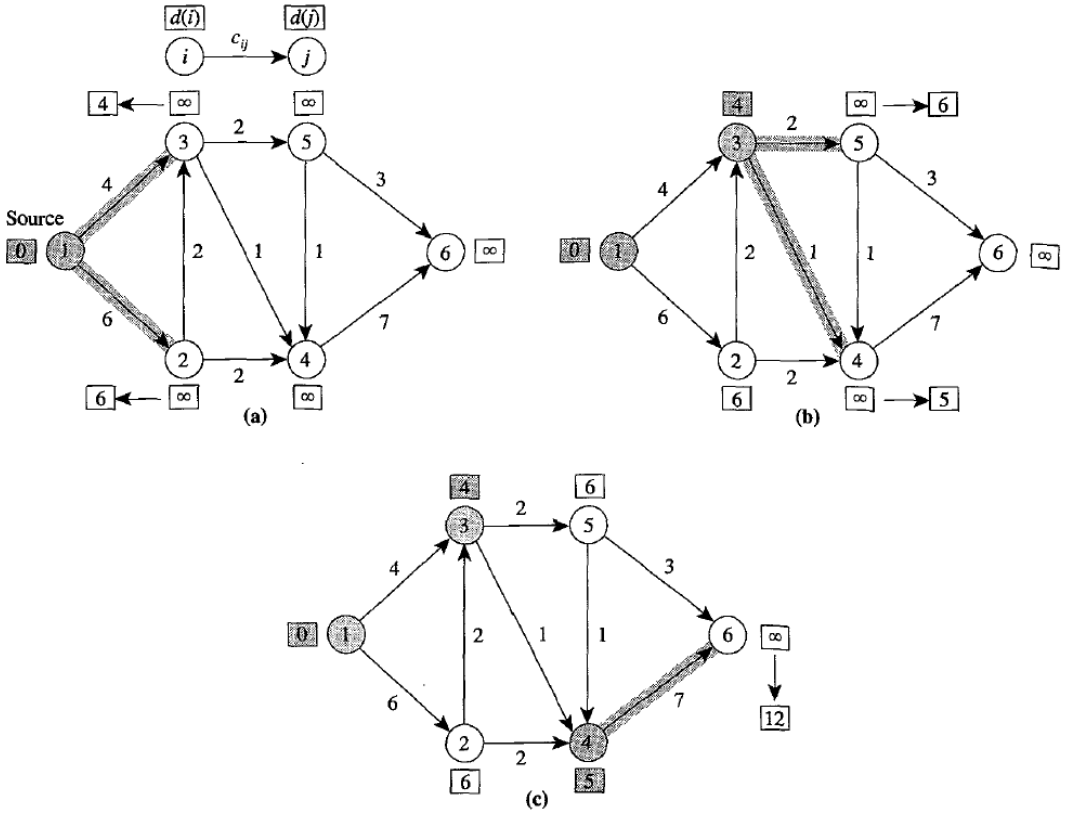


Figure 4.7 Illustrating Dijkstra's algorithm.

dijkstra 算法能解决无负网络中的最短路问题

Dijkstra 算法的运行时间

Dijkstra 算法的计算时间分为两个基本操作:

1.节点选择。该算法执行此操作 n 次,每次这样的操作都需要扫描每个临时标记的节点。因此，总的节点选择时间是 n+（n-1）~+（n-2）+…+1=0（n2）。

2.距离更新。该算法对节点 I 执行该操作|A（i）I 次。总的来说，算法执行这个操作的次数是 I A（I）I=m 次。由于每个距离更新操作需要 0（1）个时间，因此该算法需要更新所有距离标签的总时间为 O（m）。因此

定理 4.4。Dijkstra 算法在 O（n2）时间内求解最短路径问题。

节点选择和距离更新所需的时间并不平衡。节点选择总共需要 O（n2）时间，距离更新只需要 O（m）时间。因此存在改进空间

逆 Dijkstra 算法

将检查传出弧的步骤改为检查传入弧

双向 Dijkstra 算法

在最短路径问题的一些应用中，不需要确定从节点 s 到网络中其他所有节点的最短路径。假设我们要确定从节点 s 到指定节点 t 的最短路径，可以在 Dijkstra 的算法从 S'中选择 t 后立即终止它

在双向 Dijkstra 算法中，通过同时应用了节点 s 的正向 Dijkstra 算法和节点 t 的反向 Dijkstra 算法，可以更快的解决这个问题（个人理解：尽管最坏情况复杂度实际上没有改变，但是实践中双向 dijkstra 算法可能更快，因为 dijkstra 算法的永久标签往往是从节点向四周扩散，扩散速度会越来越慢，因此从首位两端同时用 dijkstra 算法能更快的找到扩散的交点位置）。

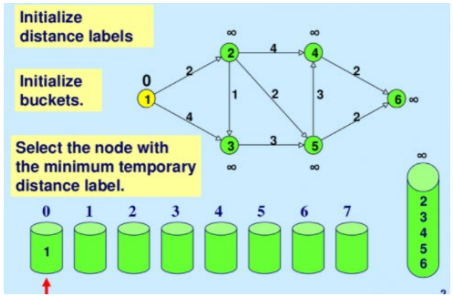
4.6dial 的实现

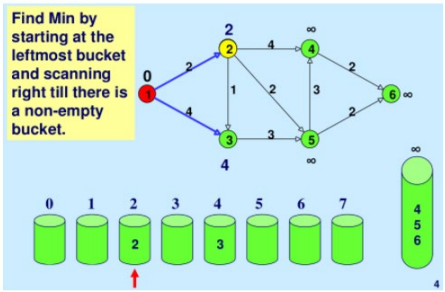
Dijkstra 算法的瓶颈操作是节点选择。

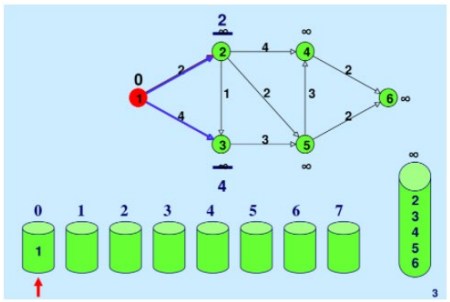
dial 算法核心：建立 nC+1 个桶，编号分别为 1，2，…，nC，C 为最大的弧长，算法大体仍采用标签设置算法的思路，但是每次生成的临时标签根据其对应的临时距离放入对应的桶中，算法从标签为 0 的桶出发，一直到编号为 nC 的桶结束

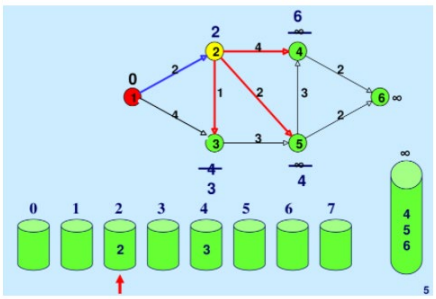
复杂度分析：桶的数据结构为双链表，此数据结构允许我们在 O（1）时间内执行以下操作：（1）检查 bucket 是空的还是非空的；（2）从 bucket 中删除元素；（3）向 bucket 中添加元素。在这种数据结构下，算法每次距离更新需要 0（1）个时间，因此所有距离更新总共需要 0（m）个时间。此实现中的瓶颈操作是在节点选择期间扫描 nC+1 存储桶。因此，Dial 算法的运行时间为 O（m+nC）。（意思就是算法需要遍历所有桶，总共需要 nC+1 次，每次更新需要遍历对应结点出发的每一条弧，虽然每个节点的弧数量不可知，但弧的总数为 m，因此 dial 算法的运行时间为 O（m+nC））

由于 Dial 的算法使用 nC+1 存储桶，它的内存需求可能会非常大。但是实际上只需要循环利用 C+1 个桶因为在算法运行过程中最大的临时标签和最小的临时标签最多差 C。









4.7 总结

原始实现

复杂度 O（n^2）

特点：

1.选择具有最小临时距离标签的节点，将其指定为永久节点，并检查与其关联的弧以修改其他距离标签。

2.很容易实现。

3.实现密集网络的最佳可用运行时间。

Dial 的实现

复杂度 O（m+nC）

特点：

1.将临时标记的节点按排序顺序存储在单位长度的 bucket 中，并通过顺序检查 bucket 来标识最小临时距离标签。

2.易于实施，具有良好的实证行为。

3.算法的运行时间是伪多项式，因此在理论上没有吸引力。

d 堆实现

复杂度 O（m $\log_d$n）d=m/n

特点：

1.使用 d-heap 数据结构来维护临时标记的节点。

2.当 m=Ω（$n^{1+\in}$），∈>0 时具有线性运行时间。

Fibonacci 堆实现

复杂度 O（m+n log n）

特点：

1.使用 Fibonacci 堆数据结构来维护临时标记节点。

2.实现了求解最短路径问题的最佳强多项式运行时间。

3.复杂且难以实施。

基数堆实现

复杂度 O（m+n log nC）

特点：

1.使用基数堆实现 Dijkstra 的算法。

2.通过将临时标记的节点存储在不同宽度的桶中，改进了 Dial 的算法。

3.对于满足相似性假设的问题，可以获得很好的运行时间。