Chapter IV Shortest Paths: Label-Setting Algorithms

1 定义与假设

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

subject to

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

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

若起终点相同

起终点不同

所作假设：1.所有弧长均为整数.
2.该网络从节点s到其他所有点均存在有向路径.
3.网络中没有负环.
4.网络是有向的.

2 最短路问题的几个应用

背包问题(转化为最大花费问题)：

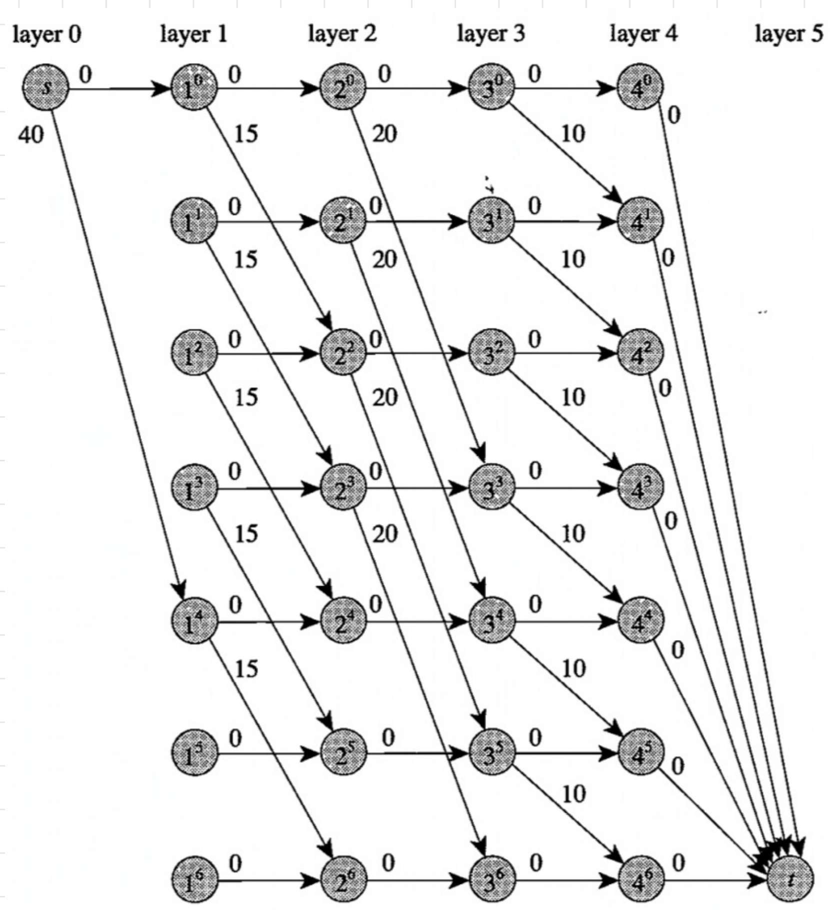


Figure 4.3 Longest path formulation of the knapsack problem.

j	1	2	3	4
u_j	40	15	20	10
w_j	4	2	3	1

通过将固定拿取组合变为流的形式，突出最短路思想。

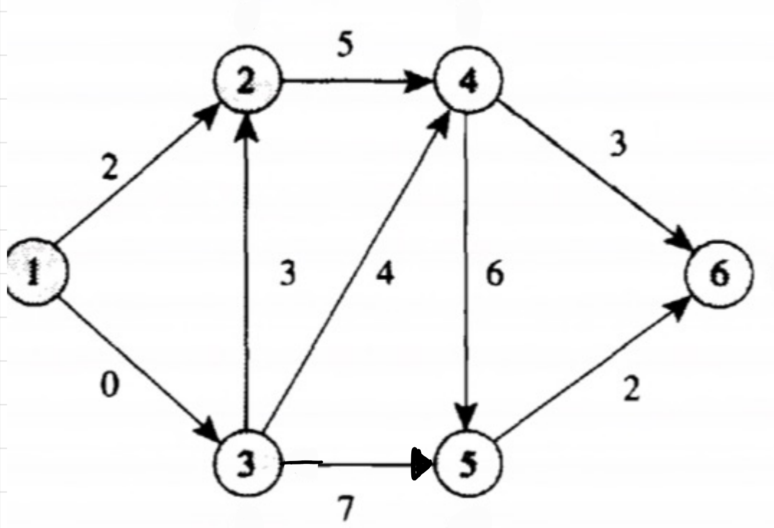
其他应用不展开赘述

3 最短路径树

性质1：如果路径s-i1-i2-···-in-k是s-k的最短路，那么对于任何q=2,3···h-1来说，路径s-i1-i2-···-is也是s-q的最短路.

性质2：如果矢量d是最短路，那么从源点到点k的有向路径P是最短路，当且仅当对于P中的所有弧，d(i)=d(j)+c(i,j).

4 无环网络中的最短路问题



无环网络最短路问题的“到达算法”

1. 设$d(s)=0$，其余点的$d(i)=M$
2. 按拓扑顺序查找点，对于点i的所有弧arc(i,j) ∈A(i)，若d(j)>d(i)+cj，则置为d(j)=d(i)+cj
3. 所有点都被检查完之后，即为最优解

5 Dijkstra算法

基本思想：

0.该算法仅适用于非负弧长的网络，其时间复杂度为O(n^2).

1.每个点都有一个距离标识d(i)，表示到点i最短距离的上限.

2.所有点分为两类：永久标号点和临时标号点，永久标号代表从原点到该点的最短路径长.

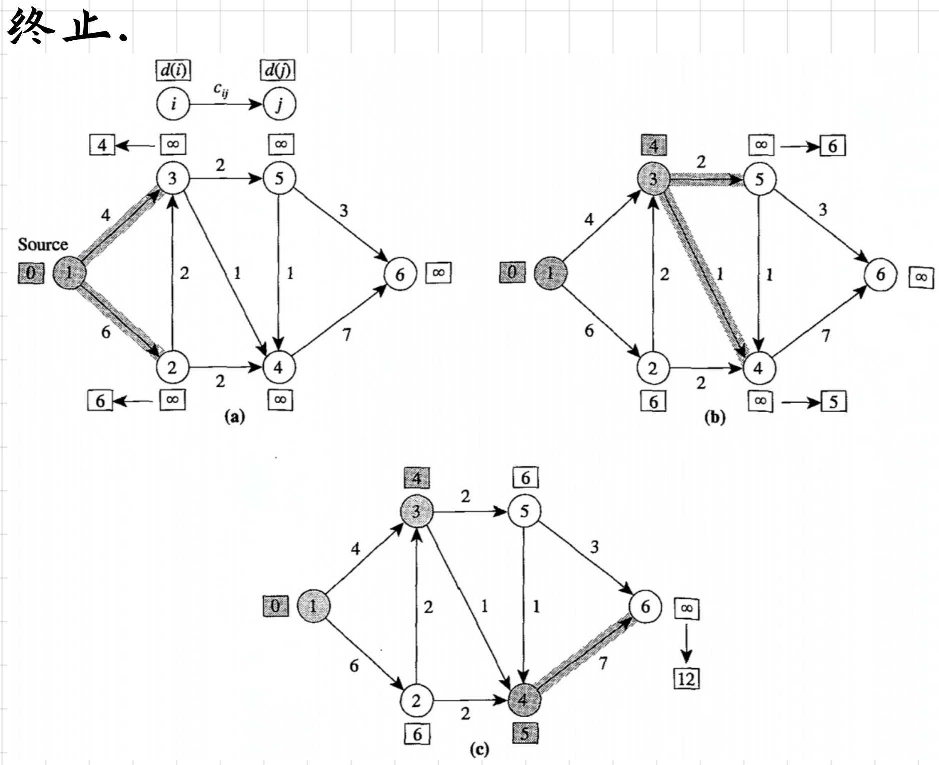
3.算法基本思路：从s与永久标号点向外展开，按照距离顺序对节点进行永久标记.

4.首先，给节点s一个永久标记0，给每个节点j一个临时标记为∞。节点i的标号就是它与源节点沿着这条路径的最短距离，该路径的内部节点(即除s或节点本身之外的节点)都被永久标记。该算法选择具有最小临时标号的节点i，使其变为永久标号，并从该节点向外延伸——也就是说，扫描A(i)中的弧来更新相邻节点的距离标号.

5.当所有节点都为永久节点时，该算法终止.

```
algorithm Dijkstra;
begin
    S : = Ø; 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) + c_ij then d(j) : = d(i) + c_ij and pred(j) : = i;
    end;
end;
```

Figure 4.6 Dijkstra's algorithm.



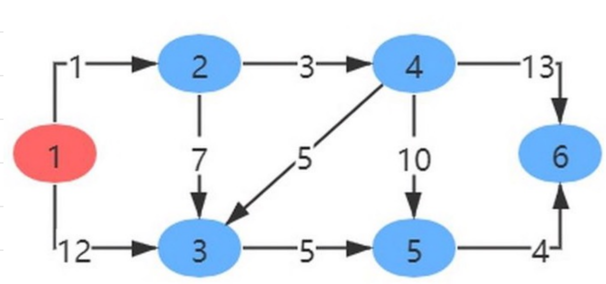
简易解释：

1.设起点标号为0，其余点为∞.

2.将标号最小的点转为永久标号；用 min(d(i)+Cij , d(j)) 更新其余点.

3.重复，直至所有点都有永久标号.

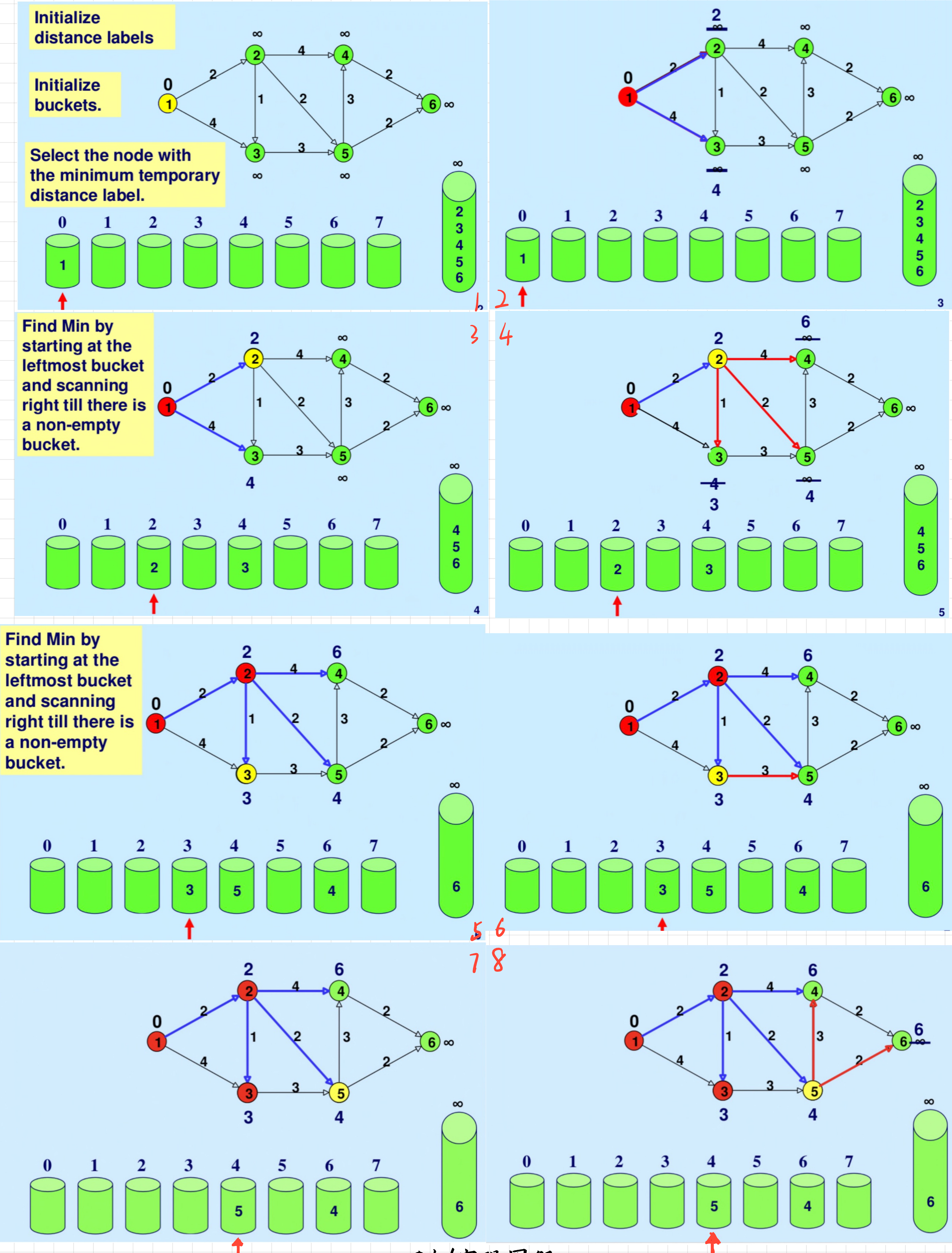
本算法所用数据结构为：邻接矩阵.



	1	2	3	4	5	6
1	0	1	12	∞	∞	∞
2	∞	0	7	3	∞	∞
3	∞	∞	0	∞	5	∞
4	∞	∞	5	0	10	13
5	∞	∞	∞	∞	0	4
6	∞	∞	∞	∞	∞	0

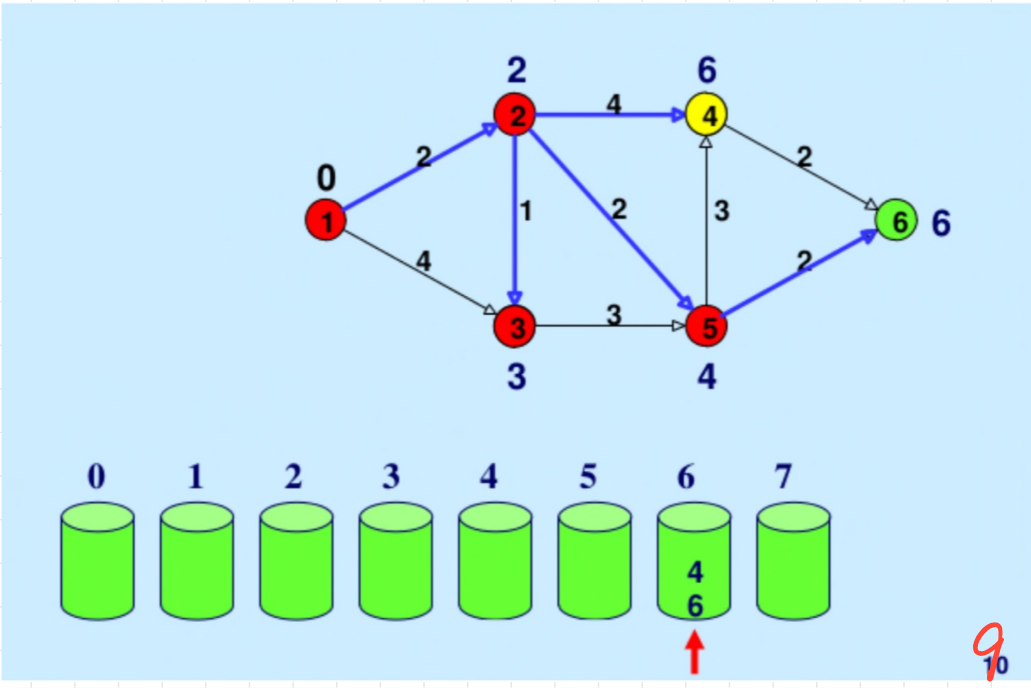
算法实现请移步GitHub查看

6 Dijkstra算法的Dial实现

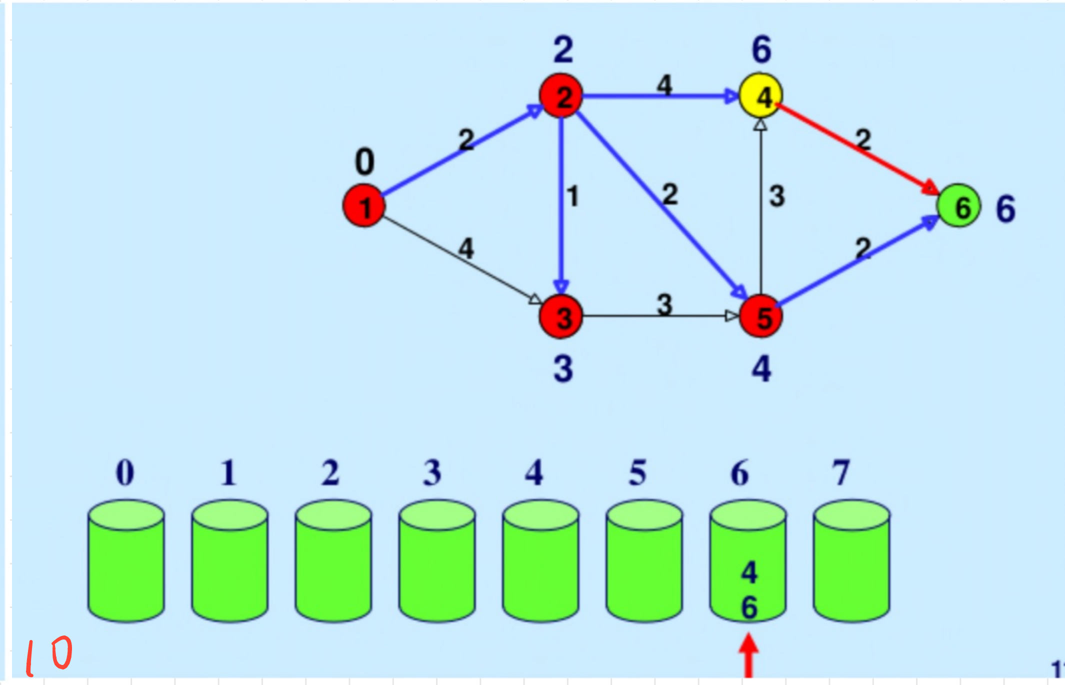


Dial实现图解

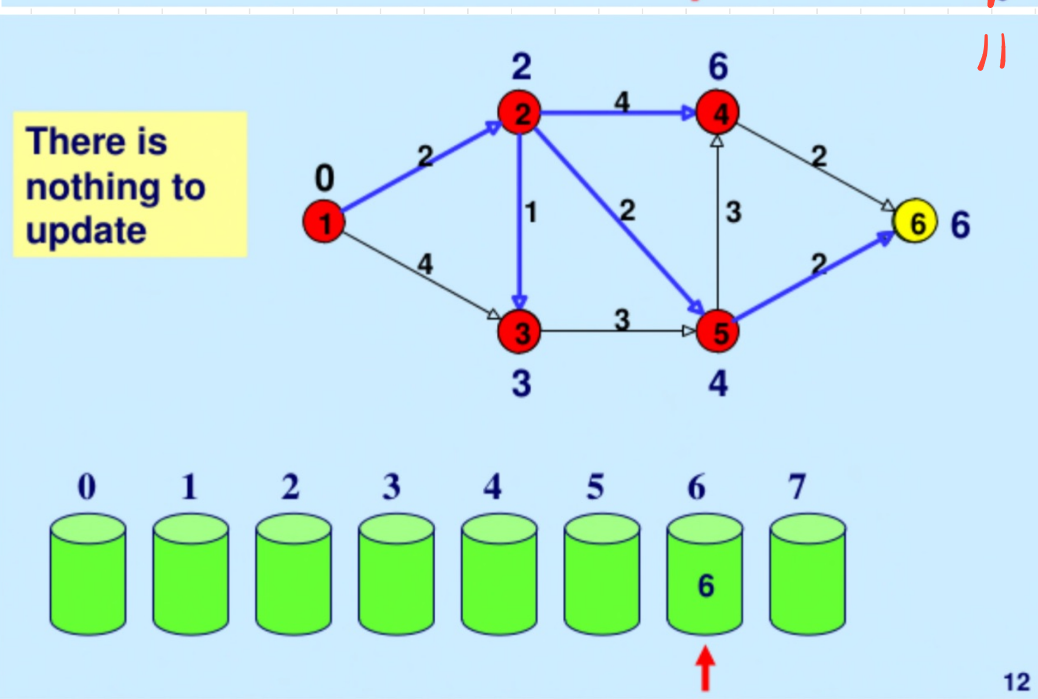
优化思路：引入“桶”(C+1个)，并从左到右扫描非空桶(扫到的永远都是最小值，从而优化寻找过程花费的时间)，将第一个扫描到的点转为永久并更新弧长与桶，直到所有桶为空。时间复杂度为O(m+nC).

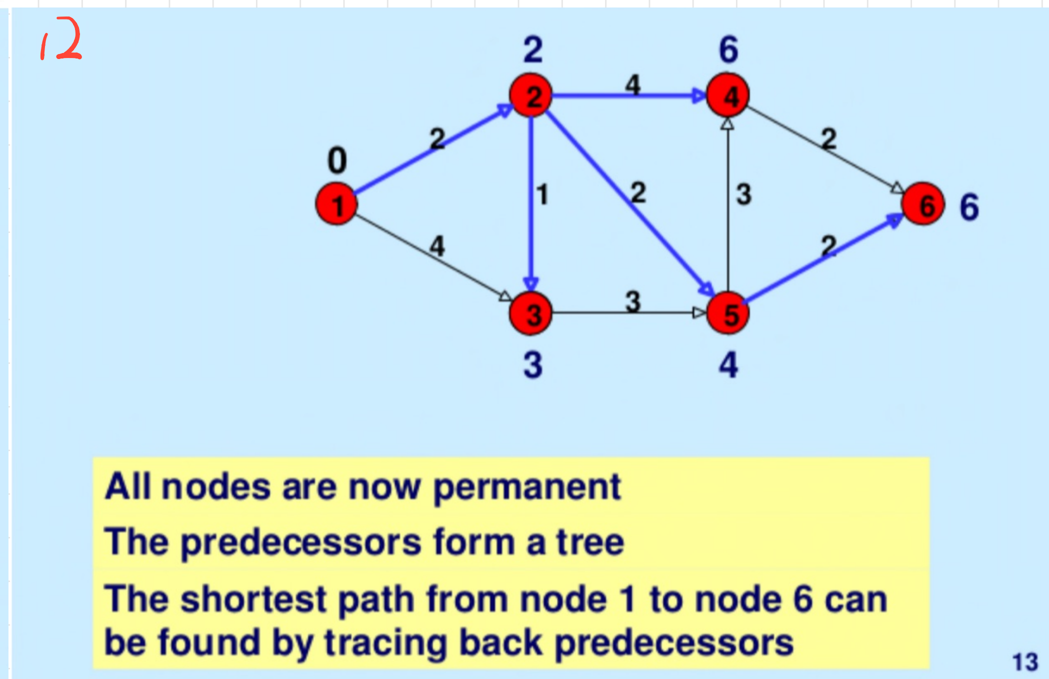


9 10



11 12





7 Dijkstra算法的堆优化

数据结构-堆：它允许我们在对象集合H上执行以下操作，每个对象都有一个称为键的相关实数。其含有以下操作：

create-heap (H). 创建一个空堆。
Find -min(i, H). 查找并返回一个键值最小的对象i。
insert(i, H). 使用预定义的键插入一个新对象i。
Reduce -key(value,i, H). 将对象i的键从当前值减少到value, value必须小于它要替换的键。
Delete -min(i, H). 删除键值最小的对象i

H将是有限临时距离标签的节点集合，节点的键将是其距离标签。

```
algorithm heap-Dijkstra;
begin
    create-heap(H);
    d(j) := ∞ for all j ∈ N;
    d(s) := 0 and pred(s) := 0;
    insert(s, H);
    while H ≠ Ø do
    begin
        find-min(i, H);
        delete-min(i, H);
        for each (i, j) ∈ A(i) do
        begin
            value := d(i) + c_ij;
            if d(j) > value then
                if d(j) = ∞ then d(j) := value, pred(j) := i, and insert (j, H)
                else set d(j) := value, pred(j) := i, and decrease-key(value, i, H);
        end;
    end;
end;
```

Figure 4.10 Dijkstra's algorithm using a heap.

其最大的特点是引入堆这一数据结构，能直接找出临时标号中的最小值，节省时间。

8 几种实现方式的对比

算法	运行时间	特性
原始实现	$O(n^2)$	1. 选择具有最小临时距离标签的节点，将其指定为永久的，并检查与之相关的弧以修改其他距离标签。 2. 非常容易实现。 3.为密集网络实现最佳可用运行时间。
表盘的实现	$O(m + nC)$	1 将临时标记的节点按排序顺序存储在单位长度的桶中，并通过依次检查桶来识别最小临时距离标签 2. 易于执行，有优秀的经验的行为。 3.该算法的运行时间是伪多项式的，因此理论上没有吸引力。
d-Heap实现	$O(m\log_d n)$其中d =m/n	1 使用d-heap数据结构来维护临时标记节点 2. 当m= 0(nl+e)对任意正e>0时的线性运行时间。
斐波那契听到实现	$O(m + n\log n)$	1. 使用斐波那契堆数据结构来维护临时标记节点。 2 给出了求解最短路径问题的最佳有效强多项式运行时间 3. li 复杂且难以实施。
基数堆实现	$O(m + n\log(nC))$	1 使用基数堆实现Dijkstra的al aorithm. 2. 改进了Dial的算法，在不同宽度的桶中存储快速标记的节点 3.为满足相似度假设的问题提供了很好的运行时间。