算法分析

刘权辉

2024 春

第四章：贪心算法

简要回顾

- 分治算法

1. 小规模易解
2. 最优子结构
3. 可合并
4. 独立性

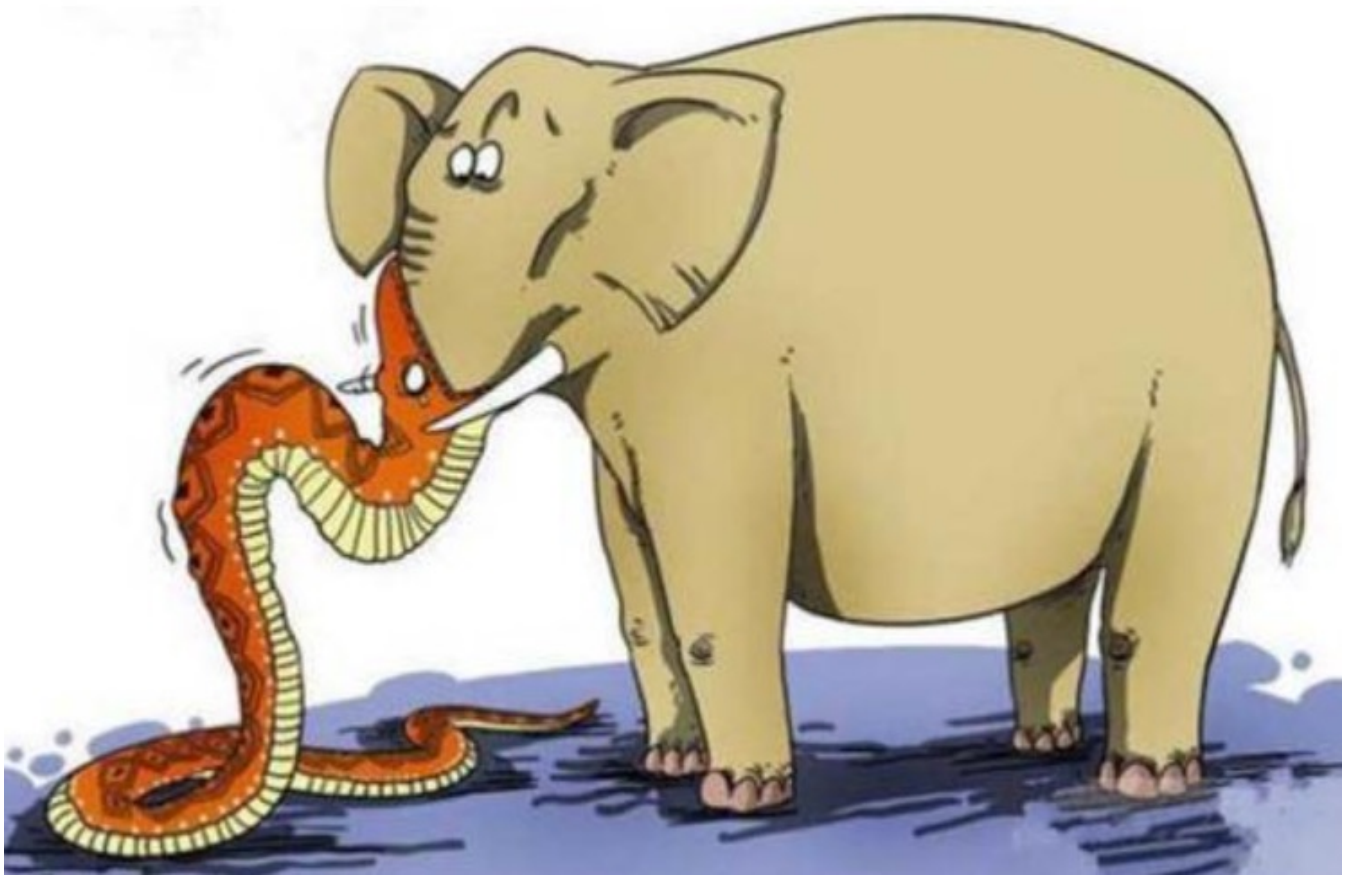
递归方程

- 动态规划

1. 最优子结构
2. 重叠子问题

自底向上

人心不足，蛇吞象

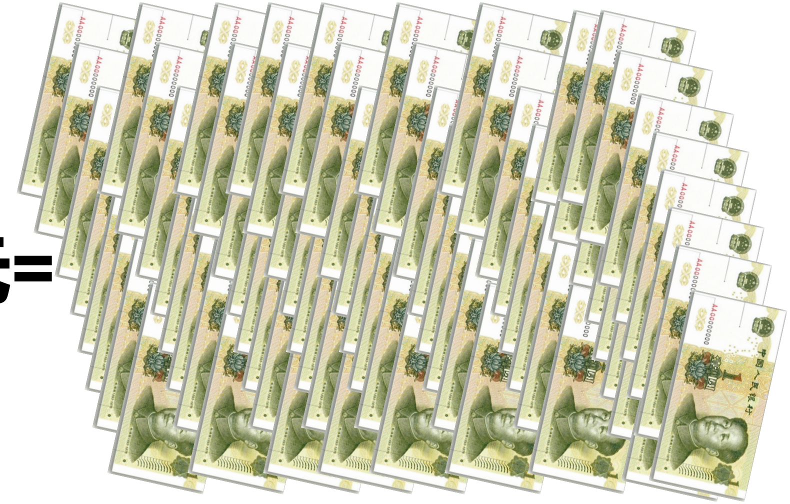


- 例：购物找零钱时，为使找回的零钱的硬币数最少，不要求找零钱的所有方案
  - 从最大面值的币种开始，按递减的顺序考虑各面额
  - 先尽量用大面值的面额，当不足大面值时才去考虑下一个较小面值，这即为贪心算法

收银员找零钱

例如一个顾客拿了一张百元的钞票买了3元钱的商品

97元=



在不超过当前需要支付零钱额度的情况下，总是选择**面额最大**的零钱

97元=

收银员算法

问题描述：从面额为 $c_1, c_2, \ldots, c_n$ (元)的零钱中，选取一些零钱，使得零钱的总额度为(x)元。

思想：在不超过当前还需支付零钱额度的情况下，总是选取面额最大的零钱。

Cashiers_Algorthm(x, c_1,..., c_n)

Sort $0<c_1<c_2<\ldots<c_n$.

S={}.

While (x>0)

  k ← largest c_k such that $c_k \le$ x.

  If no such k, return “error”.

  Else

    x ← x - c_k.

    S ← S ∪ c_k.

Return S

收银员算法

Cashiers_Algorthm$(x, c_1, c_2, \ldots, c_n)$

Sort $0 < c_1 < c_2 < \ldots < c_n$.

S={}.

While (x>0)

  k ← largest c_k such that $c_k \le$ x.

  If no such k, **return** "error".

  Else

    x ← x - c_k.

    S ← S ∪ c_k.

Return S

c_1 c_2 c_3 c_4 c_5

初始：
- x=97
- S={}

第1次: x=97

判断: x > 0？是

遍历: c_5=50 ≤ x

更新: x=x-c_5=47 (还需找零)

S = 

收银员算法

Cashiers_Algorthm$(x, c_1, c_2, \dots, c_n)$

Sort $0<c_1<c_2<\dots<c_n$.

S={}.

While (x>0)

  k ← largest c_k such that $c_k \le x$.

  If no such k, **return** “error”.

  Else

    $x \leftarrow x - c_k$.

    $S \leftarrow S \cup c_k$.

Return S

c_1 c_2 c_3 c_4 c_5

第2次: x=47

判断: x > 0 ? 是

遍历: c_4=20 ≤ x

更新: x=x-c_4=27 (还需找零)

S =

收银员算法

Cashiers_Algorthm$(x, c_1, c_2, \ldots, c_n)$

Sort $0 < c_1 < c_2 < \ldots < c_n$.

$S = \{\}$.

While $(x > 0)$

$\quad k \leftarrow$ largest c_k such that $c_k \le x$.

$\quad$ **If** no such k, **return** "error".

$\quad$ **Else**

$\quad\quad x \leftarrow x - c_k$.

$\quad\quad S \leftarrow S \cup c_k$.

Return S

$c_1 \quad c_2 \quad c_3 \quad c_4 \quad c_5$

第3次: x=27

判断: x > 0 ? 是

遍历: c_4=20 ≤ x

更新: x=x-c_4=7 (还需找零)

S =

收银员算法

Cashiers_Algorthm($x, c_1, c_2, \ldots, c_n$)

Sort $0<c_1<c_2<\ldots<c_n$.

S={}.

While (x>0)

  k ← largest c_k such that $c_k \le$ x.

  If no such k, **return** "error".

  Else

    x ← x - c_k.

    S ← S ∪ c_k.

Return S

第4次: x=7

判断: x > 0 ? 是

遍历: c_2=5 ≤ x

更新: x=x-c_2=2 (还需找零)

S =

c_1 c_2 c_3 c_4 c_5

收银员算法

Cashiers_Algorthm$(x, c_1, c_2, \ldots, c_n)$

Sort $0<c_1<c_2<\ldots<c_n$.

S={}.

While (x>0)

  k ← largest c_k such that $c_k \le$ x.

  If no such k, **return** "error".

  Else

    x ← x - c_k.

    S ← S ∪ c_k.

Return S

c_1 c_2 c_3 c_4 c_5

第5次: x=2

判断: x > 0 ? 是

遍历: c_1=1 ≤ x

更新: x=x-c_1=1 (还需找零)

S =

收银员算法

Cashiers_Algorthm$(x, c_1, c_2, \ldots, c_n)$

Sort $0 < c_1 < c_2 < \ldots < c_n$.

S={}.

While (x>0)

  k ← largest c_k such that $c_k \le$ x.

  If no such k, **return** "error".

  Else

    x ← x - c_k.

    S ← S ∪ c_k.

Return S

c_1 c_2 c_3 c_4 c_5

第6次: x=1

判断: x > 0 ? 是

遍历: c_1=1 ≤ x

更新: x=x-c_1=0 (还需找零)

S =

收银员算法

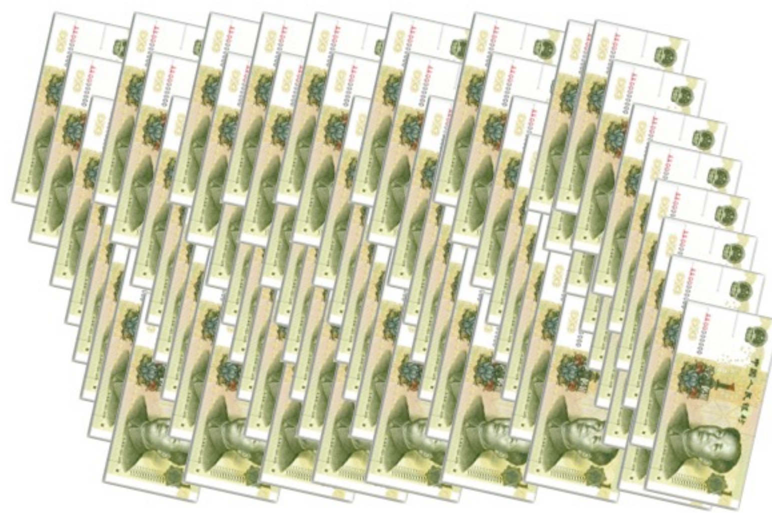
Cashiers_Algorthm($x, c_1, c_2, \ldots, c_n$)

Sort $0 < c_1 < c_2 < \ldots < c_n$.
S={}.
While (x>0)
    k ← largest c_k such that $c_k \le x$.
    If no such k, return “error”.
    Else
        $x \leftarrow x - c_k$.
        $S \leftarrow S \cup c_k$.
Return S

第7次: x=0

判断: x > 0 ? 否

跳出循环，返回 S



c_1 c_2 c_3 c_4 c_5

收银员算法分析

收银员找零的过程中，总是选取面额最大且不超过当前所付的额度。直观上，这种方法找零每用一张零钱，剩下还需付给顾客的零钱最少。也就是说，该算法贪心选择的意义是每用一张零钱，使得当前支付给顾客的零钱最多，从而达到支付给顾零钱的总张数最少的目的。

问题

收银员算法是最优的吗?

收银员算法总是最优的

只要c_1=1, 对任意 $c_1 < c_2 < \cdots < c_n$, 收银员算法是最优的

Q：收银员算法总是最优的?

不是　　　　8张

- 收银员算法:140¢ = 100 + 34 + 1 + 1 + 1 + 1 + 1 + 1
- 最优算法：140¢ = 70 + 70　2张

1　10　21　34　70　100　350　1225　1500

邮资种类(美分)

不是,甚至没有可行解，如果 $c_1 > 1$：7，8，9

- 收银员算法：15 = 9 + ?
- 最优算法：15 = 7 + 8

学习要点

- 理解贪心算法的概念。
- 掌握贪心算法的基本要素：
  - 最优子结构性质
  - 贪心选择性质
- 理解贪心算法与动态规划算法的差异。
- 理解贪心算法的一般理论。
- 通过应用范例学习贪心算法设计策略：
  - 活动安排问题；
  - 最优装载问题；
  - 哈夫曼编码；
  - 单源点最短路径；
  - 最小生成树；
  - 多机调度

引言

- 贪心算法总是作出在当前看来最好的选择。也就是说贪心算法并不从整体最优考虑，它所作出的选择只是在某种意义上的局部最优选择。

- 当然，希望贪心算法得到的最终结果也是整体最优的。

- 虽然贪心算法不能对所有问题都得到整体最优解，但对许多问题它能产生整体最优解。如单源最短路经问题，最小生成树问题等。

- 在一些情况下，即使贪心算法不能得到整体最优解，其最终结果却是最优解的很好近似。

4.1贪心算法的基本要素

贪心算法的基本要素

- 讨论可以用贪心算法求解的问题的一般特征

- 对于一个具体的问题，怎么知道是否可用贪心算法解此问题，以及能否得到问题的最优解呢？没有标准答案
- 但是，从许多用贪心算法求解的问题中看到这类问题一般具有2个重要的性质：贪心选择性质和最优子结构性质。

贪心选择性质

- 所谓贪心选择性质是指所求问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来达到。这是贪心算法可行的第一个基本要素，也是贪心算法与动态规划算法的主要区别。

- 动态规划算法通常以自底向上的方式解各子问题，而贪心算法则通常以自顶向下的方式进行，以迭代的方式作出相继的贪心选择，每作一次贪心选择就将所求问题简化为规模更小的子问题。

- 对于一个具体问题，要确定它是否具有贪心选择性质，必须证明每一步所作的贪心选择最终导致问题的整体最优解。

最优子结构性质

- 当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质。
- 问题的最优子结构性质是该问题可用动态规划算法或贪心算法求解的关键特征。

贪心算法与动态规划算法的差异

- 贪心算法和动态规划算法都要求问题具有最优子结构性质，这是2类算法的一个共同点。

- 但是，对于具有最优子结构的问题应该选用贪心算法还是动态规划算法求解?

- 是否能用动态规划算法求解的问题也能用贪心算法求解?

- 下面研究2个经典的组合优化问题，并以此说明贪心算法与动态规划算法的主要差别。

问题1：0-1背包问题

- 给定n种物品和一个背包，物品i的重量是w_i，其价值为v_i，背包的容量为C。应如何选择装入背包的物品，使得装入背包中物品的总价值最大?

> ➢ 在选择装入背包的物品时，对每种物品i只有2种选择，即装入背包或不装入背包。不能将物品i装入背包多次，也不能只装入部分的物品i。

问题2：背包问题

- 与0-1背包问题类似，所不同的是在选择物品i装入背包时，可以选择物品i的一部分，而不一定要全部装入背包，1≤i≤n。

> - 这2类背包问题都具有最优子结构性质，极为相似。但背包问题可以用贪心算法求解，而0-1背包问题却不能用贪心算法求解。
> - 为什么？

贪心法解背包问题的基本步骤

- 首先，计算每种物品单位重量的价值V_i/W_i;
- 然后，依贪心选择策略，将尽可能多的单位重量价值最高的物品装入背包: 若将这种物品全部装入背包后，背包内的物品总重量未超过C，则选择单位重量价值次高的物品并尽可能多地装入背包；
- 依此策略一直地进行下去，直到背包装满为止。

具体算法可描述如下页：

贪心法解背包问题的基本步骤

```
void Knapsack(int n,float M,float v[],float w[],float x[]){
    Sort(n,v,w);
    int i;
    for (i=1;i<=n;i++) x[i]=0;
    float c=M;
    for (i=1;i<=n;i++) {
        if (w[i]>c) break;
        x[i]=1;
        c-=w[i]; }
    if (i<=n)    x[i]=c/w[i];
}
```

- 算法knapsack的主要计算时间在于将各种物品依其单位重量的价值从大到小排序。因此，算法的计算时间上界为O（nlogn）。
- 为了证明算法的正确性，还必须证明背包问题具有贪心选择性质。

Data Intelligence & Computing Art Laboratory

分析

- 对于0-1背包问题，贪心选择之所以不能得到最优解是因为在这种情况下，它无法保证最终能将背包装满，部分闲置的背包空间使每公斤背包空间的价值降低了。
- 事实上，在考虑0-1背包问题时，应比较选择该物品和不选择该物品所导致的最终方案，然后再作出最好选择。由此就导出许多互相重叠的子问题。这正是该问题可用动态规划算法求解的另一重要特征。
- 实际上也是如此，动态规划算法的确可以有效地解0-1背包问题。

贪心法解背包问题

假设背包容量：M=5 kg，三个物品重量与总价值为：

x1=(1kg,5元)；

x2=(2kg,9元)；

x3=(3kg,12元)；

针对0-1贪心问题，按照单位重量价值从大到小排序：只能将x1和x2放入背包里面，但显然将物品x2和x3放入背包总价值最大。

4.2活动安排问题

活动安排问题

- **问题描述**：
  - 设有n个活动的集合E={1,2,…,n}，其中每个活动都要求使用同一资源，如演讲会场等，而在同一时间内只有一个活动能使用这一资源。
  - 每个活动i都有一个要求使用该资源的起始时间s_i和一个结束时间f_i，且 $s_i < f_i$ 。如果选择了活动i，则它在半开时间区间$[s_i, f_i)$内占用资源。
  - 若区间 $[s_i, f_i)$ 与区间 $[s_j, f_j)$ 不相交，则称活动i与活动j是相容的。也就是说，当 $s_i \geq f_j$ 或 $s_j \geq f_i$ 时，活动i与活动j相容。

在所给的活动集合中选出最大的相容活动子集合

- 例：设待安排的11个活动的开始时间和结束时间按结束时间的非减序排列如下：

i	1	2	3	4	5	6	7	8	9	10	11
S[i]	1	3	0	5	3	5	6	8	8	2	12
f[i]	4	5	6	7	8	9	10	11	12	13	14

活动安排问题 $s_j \geq f_i$

i	1	2	3	4	5	6	7	8	9	10	11
S[i]	1	3	0	5	3	5	6	8	8	2	12
f[i]	4	5	6	7	8	9	10	11	12	13	14

分析：

- 目标是要选择尽可能多的活动！--先选择结束时间最早的那个活动，以便于可以安排更多的活动。按这种方法选择相容活动为未安排活动留下尽可能多的时间。
- 也就是说，该算法的贪心选择的意义是使剩余的可安排时间段极大化，以便安排尽可能多的相容活动。
- 然后选择下一个活动。若被检查的活动i的开始时间s_i小于最近选择的活动j的结束时间f_j，则不选择活动i，否则选择活动i。

活动安排问题 $s_j \geq f_i$

```
template<class Type>
void GreedySelector(int n, Type s[], Type f[], bool A[]){
    A[1]=true;
    int i=1;
    for (int j=2;j<=n;j++) {
        if (s[j]>=f[i]) {
            A[j]=true; i=j;
        }
        else A[i]=false;
    }
}
```

- 算法GreedySelector的效率极高。
- 当输入的活动已按结束时间的非减序排列，算法只需O(n)的时间安排n个活动，使最多的活动能相容地使用公共资源。
- 如果所给出的活动未按非减序排列，可以用O(nlogn)的时间重排。

活动安排问题

□讨论：贪心算法并不总能求得问题的最优解。但是，对于活动安排问题，贪心算法却总能求得整体最优解，即最终确定的相容集合规模最大。

数学归纳法证明：

- 设E={1，2，…，n}为所给的活动集合，并已按结束时间升序排序，则活动1具有最早完成时间。
- 首先，需证明活动安排问题有一个最优解以贪心选择开始，即最优解包括活动1。
- （证明见下页）

贪心选择性质证明（续）：

- 设A ⊆ E是该问题的一个最优解，且A中活动已排序，其中第一个活动为k。那么：

  1）若k=1，则A就是一个以贪心选择开始的最优解;

  2）若k>1，则设B={A-{k}} ∪{1}, 由于$f_1 \le f_k$，且A中活动相容，则B中也相容。而B中活动个数等于A中活动个数。又A最优，所以B也最优。

  即：B是一个以贪心选择活动1开始的最优活动安排。

- 因此，总存在一个以贪心选择开始的最优活动安排方案, 具有贪心选择性质

最优子结构性质证明（续）：

- 在做了贪心选择，选择了活动1之后，原问题即简化为对E中所有与活动1相容的活动进行安排的子问题；
- 即：若A是原问题的一个最优解，则A′=A-{1}是活动安排子问题E′={ i∈E:$s_i \geq f_1$ } 的一个最优解；
- （反证法证明）假如若能找到E′的一个解B′，它包含比A′更多的活动，则将活动1加入到B′中将产生E的一个解B，他包含比A更多的活动，这与 A 是最优解相矛盾；
- 因此，每一步所做的贪心选择都将问题简化为一个更小的与原问题相同性质的子问题，即具有最优子结构性质；

活动安排问题具有贪心选择性质和最优子结构性质。

4.3 最优装载问题

最优装载问题

□ 问题描述：

- 有一批集装箱要装上一艘载重量为c的轮船。其中集装箱i的重量为w_i；
- 最优装载问题要求确定在装载体积不受限制的情况下，将尽可能多的集装箱装上轮船。形式化描述如下：

$$\max \sum_{i=1}^{n} x_i$$

$$\sum_{i=1}^{n} w_i x_i \le c$$

$$x_i \in \{0,1\},\ 1 \le i \le n$$

分析：

- 最优装载问题可用贪心算法求解，采用重量最轻者先装的贪心选择策略，可产生最优装载问题的最优解。

最优装载问题：贪心选择性质证明

- 设集装箱已按重量大小进行升序排序，且（$x_1, x_2, \ldots, x_n$）是最优装载问题的一个最优解，即0或1的序列;
- 设 k=min{ i | x_i=1}，其中1≤i≤n，如果给定的最优装载问题有解，则1≤k≤n:

  1）当k=1时，$(x_1, x_2, \ldots, x_n)$是一个满足贪心选择性质的最优解；

  2）当k>1时，取$y_1=1$，$y_k=0$，$y_i=x_i$，1<i≤n，i≠k，则：

$$\sum_{i=1}^{n} w_i y_i = w_1 - w_k + \sum_{i=1}^{n} w_i x_i \le \sum_{i=1}^{n} w_i x_i \le C$$

- 因此，（$y_1, y_2, \ldots, y_n$）是所给问题的可行解;
- 另一方面，由 $\sum_{i=1}^{n} y_i = \sum_{i=1}^{n} x_i$ 知：（$y_1, y_2, \ldots, y_n$）是满足贪心选择性质的最优解。

最优子结构性质

证明：

- 设$(x_1, \ldots, x_n)$是最优装载问题满足贪心选择性质的最优解，则易知$x_1 = 1, (x_2, \ldots, x_n)$是轮船载重量为$c - w_1$，待装船集装箱$\{2,3, \ldots, n\}$时相应最优装载问题的最优解。即，最优装载问题满足最优子结构性质。

时间复杂性：

- 最优装载问题的主要计算量在于集装箱的排序，故算法时间复杂度为$O(nlogn)$

最优装载问题：算法描述

```
template<class Type>
void Loading(int x[],  Type w[], Type c, int n){
  int *t = new int [n+1];
  Sort(w, t, n);
  for (int i = 1; i <= n; i++)      x[i] = 0;
  for (int i = 1; i <= n && w[t[i]] <= c; i++) {
        x[t[i]] = 1;
        c -= w[t[i]];}
}
```

4.4 哈夫曼编码

□ 哈夫曼编码

哈夫曼编码是广泛地用于数据文件压缩的十分有效的编码方法。该方法完全依据字符出现概率来构造异字头的平均长度最短的码字。给出现频率高的字符较短的编码，出现频率较低的字符以较长的编码，可以大大缩短总码长。

哈夫曼编码示例

- 一个文件有100000个字符，有6个不同的字符，字符出现频率，定长编码及边长编码如下表4-1所示：

	a	b	c	d	e	f
频率(千次)	45	13	12	16	9	5
定长码	000	001	010	011	100	101
变长码	0	101	100	111	1101	1100

编码后长度， 定长码：300000位，变长码：224000位

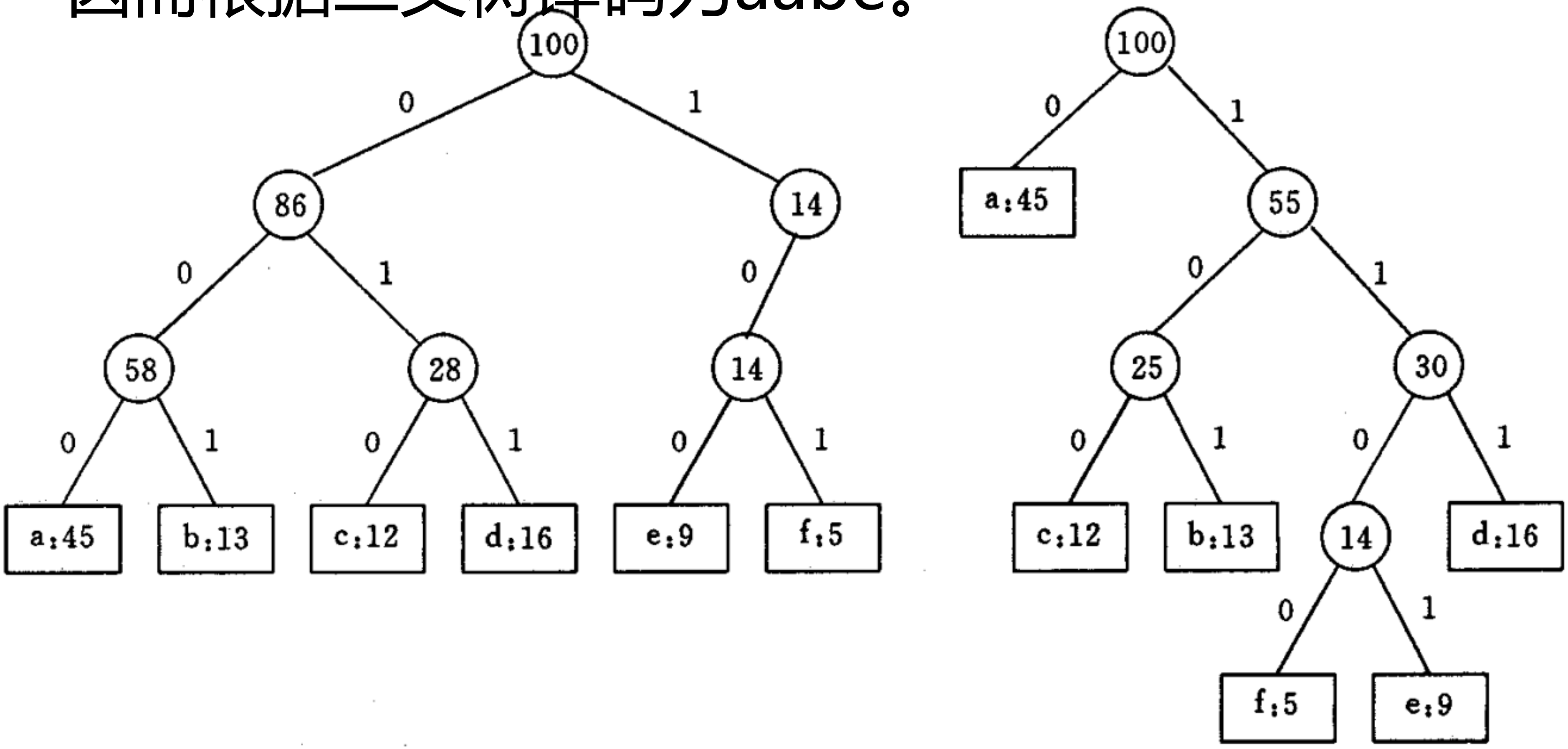
□ 前缀码

- 对每一个字符规定一个0,1串作为其代码，并要求任一字符的代码都不是其它字符代码的前缀，这种编码称为前缀码;

- 由于任一字符的代码都不是其他字符代码的前缀，从编码文件中不断根据最优前缀码二叉树取出代表某一字符的前缀码，转换即可。

哈夫曼编码

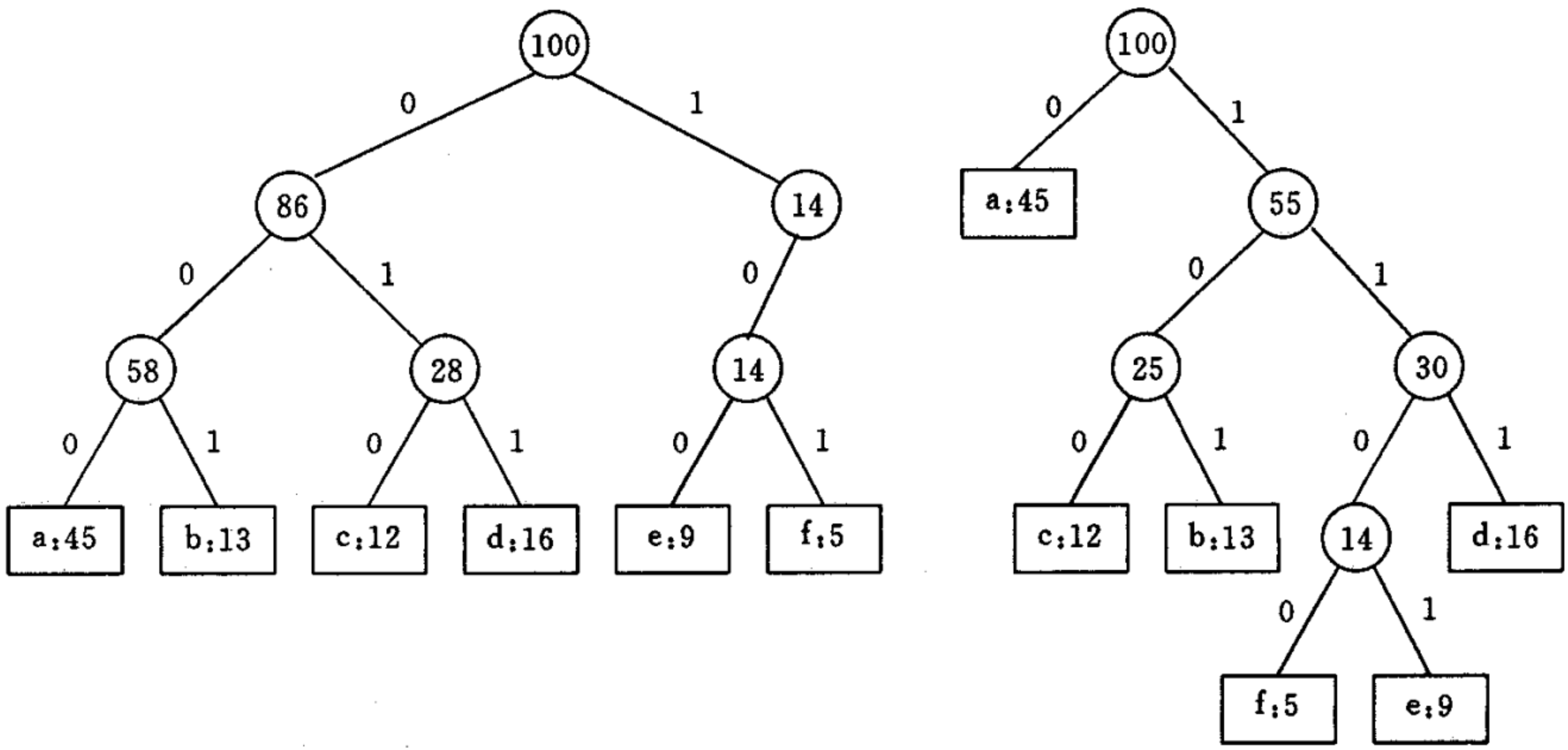
- 例，表中的变长码就是一种前缀码。对于给定的0,1串001011101可唯一地分解为0，0，101，1101，因而根据二叉树译码为aabe。



(a) **定长码** (b) **变长码**

哈夫曼编码

平均编码长度



(a)　　(b)

□ 哈夫曼编码构造

- 哈夫曼编码可用贪心算法构造
- 构造过程：自底向上构造哈夫曼树

算法描述：

- 算法以|C|个叶结点开始，执行|C|-1次的“合并”运算后产生最终所要求的树T；
- 设编码字符集中每一字符c的频率是f(c)；
- 以f为键值的优先队列Q 用在做贪心选择时，有效地确定算法当前要合并的两棵具有最小频率的树；
- 一旦两棵具有最小频率的树合并后，产生一棵新的树，其频率为合并的两棵树的频率之和，并将新树插入优先队列Q。

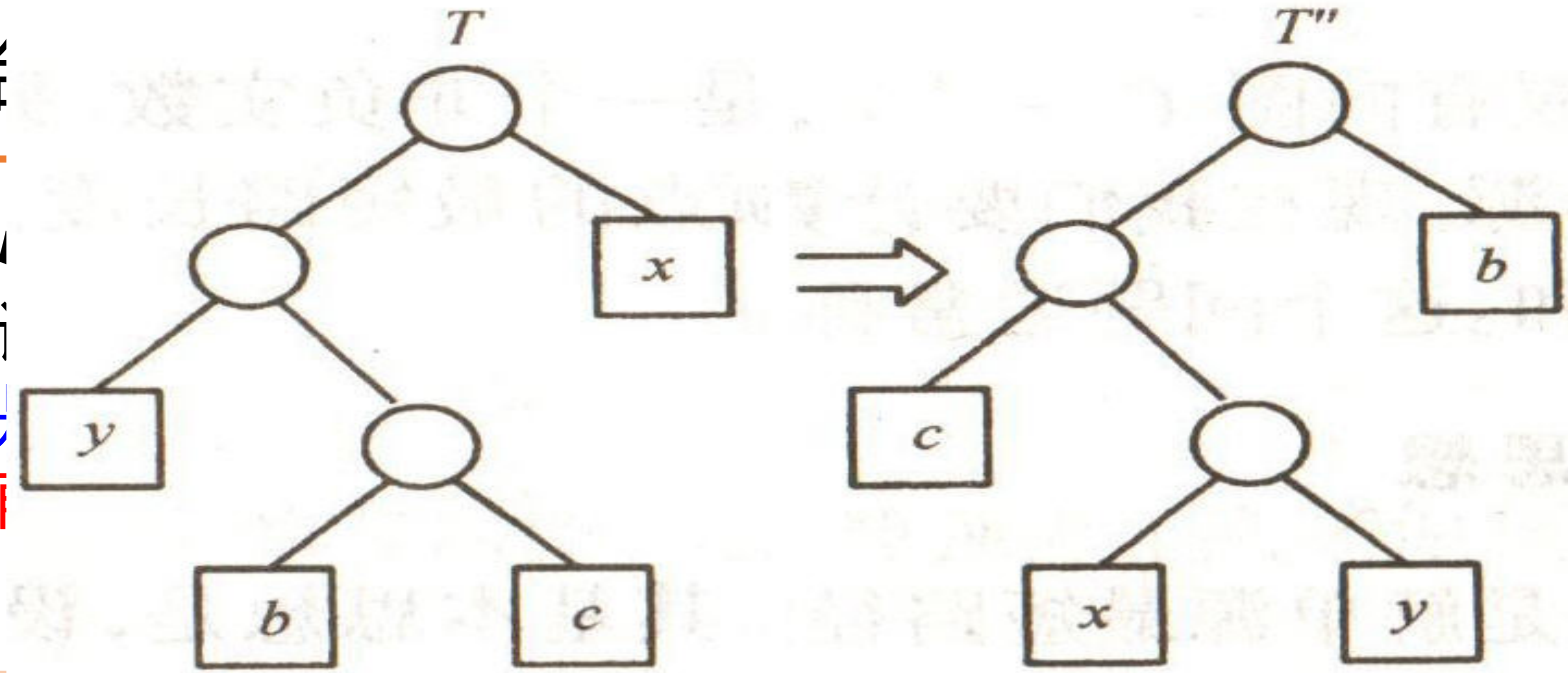


图 4-5 编码树 T 的变换

证明：

- 设二叉树T表示C的任意一个最优前缀码;
- 我们要证明可以对T作适当修改后得到一颗新的二叉树T″，使得在新树中 x 和 y 是最深叶子且为兄弟;
- 同时新树 T″ 表示的前缀码也是C的一个最优前缀码。如果我们能做到这一点，则 x 和 y 在 T″ 表示的最优前缀码中就具有相同的码长且最后一位编码不同。

哈夫曼编码

证明：设b和c是T的最深叶子且为兄弟。不失一般性，可设f(b) ≤f(c)，f(x) ≤f(y)；由于x和y是C中具有最小频率的两个字符，故 f(x) ≤f(b)，f(y) ≤f(c)；

➢ 我们首先在树T中交换叶子b和x的位置得到树T'，然后在树T'中再交换叶子c和y的位置，得到树T''。如下图示：

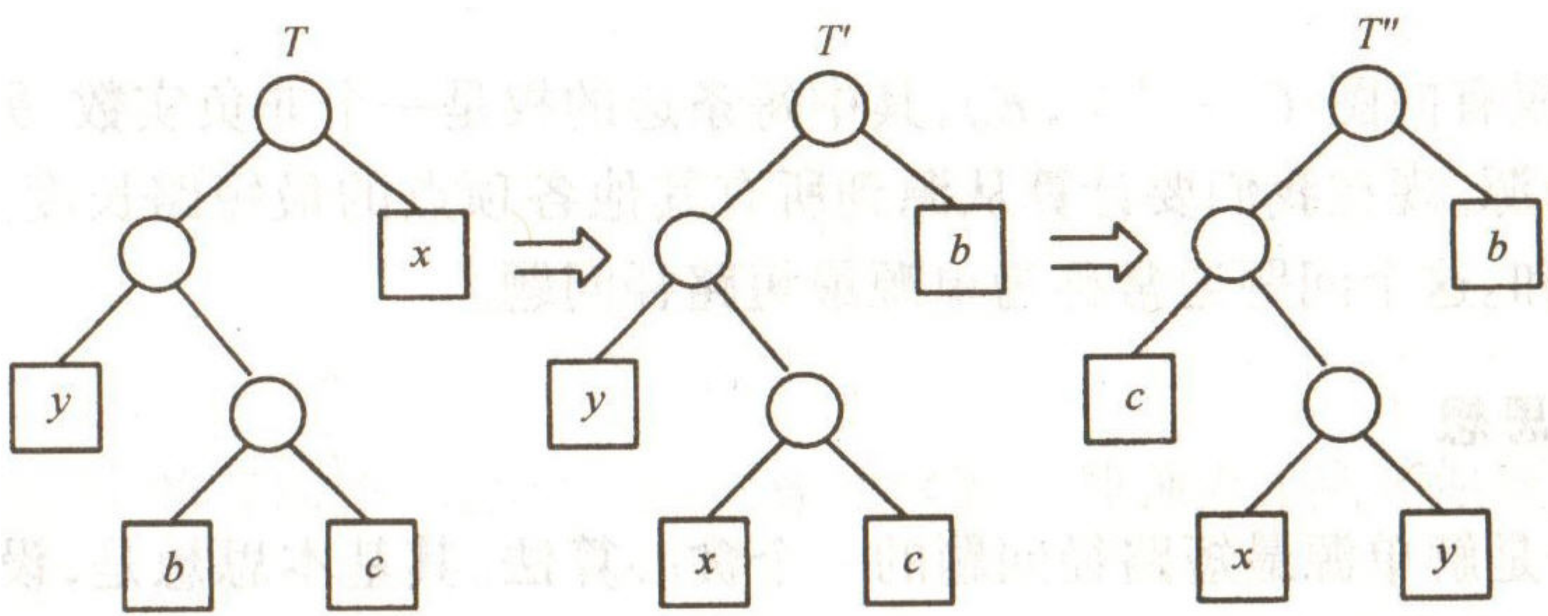


图 4-5　编码树 *T* 的变换

哈夫

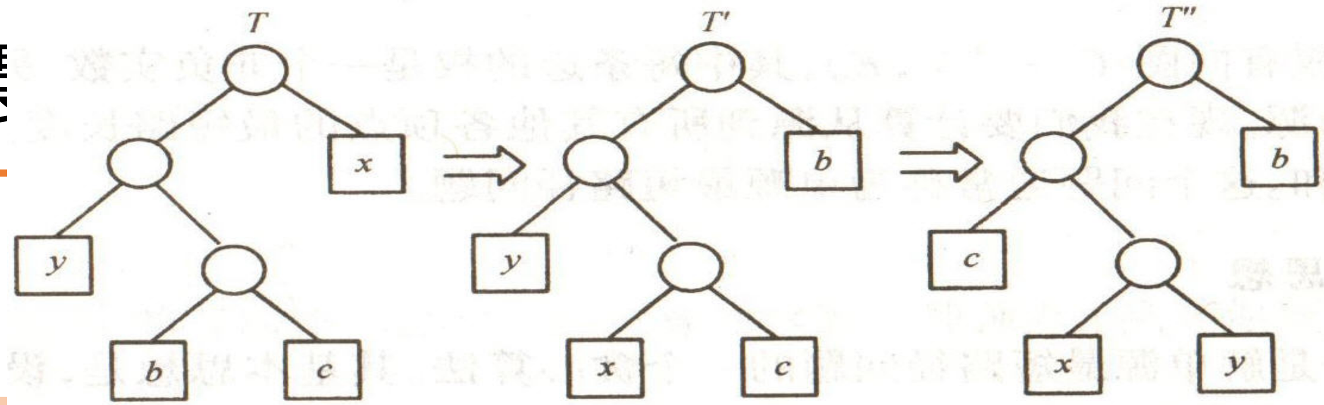


图 4-5 编码树 T 的变换

证明（续）

➢ 由此可以知道，树T和T’表示的前缀码的平均码长之差为：

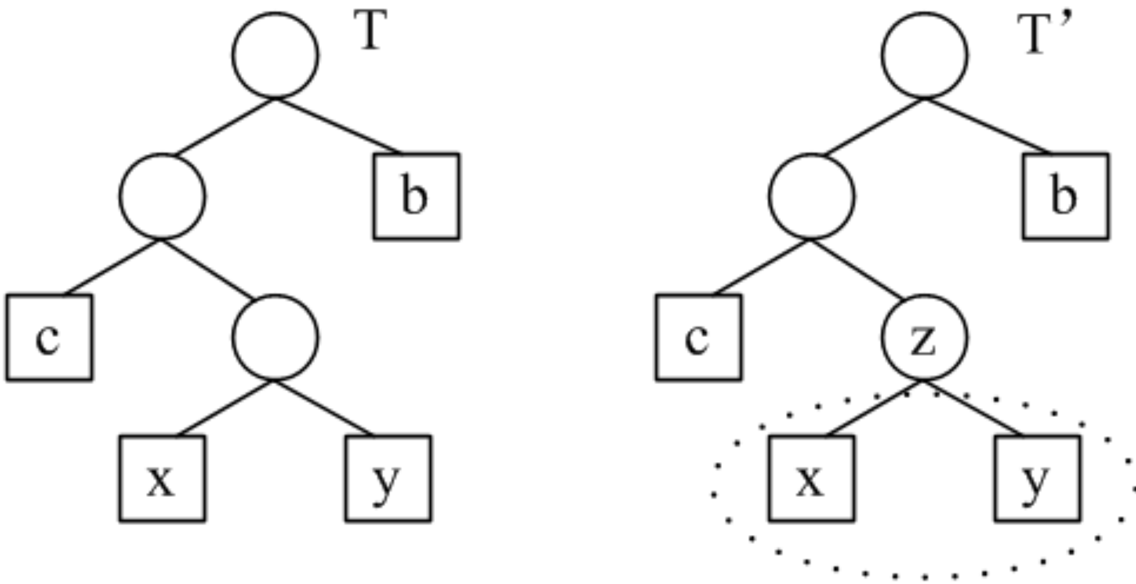
$$
\begin{aligned}
B(T)-B(T') &= \sum_{c\in C} f(c)d_T(c) - \sum_{c\in C} f(c)d_{T'}(c) \\
&= f(x)d_T(x)+f(b)d_T(b)-f(x)d_{T'}(x)-f(b)d_{T'}(b) \\
&= f(x)d_T(x)+f(b)d_T(b)-f(x)d_T(b)-f(b)d_T(x) \\
&= (f(b)-f(x))(d_T(b)-d_T(x)) \geqslant 0
\end{aligned}
$$

➢ 同理可知： B(T’)- B(T”)≥0, 因此，B(T”) ≤ B(T’) ≤ B(T)。

➢ 另一方面，由于T所表示的前缀码是最优的，故：
B(T) ≤ B(T”), 从而可知：B(T) = B(T”)，即T”表示的前缀码也是最优的前缀码，且x和y具有最长的码长，同时仅最后一位编码不同。

□ 最优子结构性质

设T是表示字符集C的一个最优前缀码的完全二叉树，C中字符c的出现频率为f(c)。设x和y是树T中的两个叶子且为兄弟，z是它们的父亲。若将z看成是具有频率f(z)=f(x)+f(y)的字符，则树 T’ =T-{x，y} 表示字符集C’ ={ C-{x，y}} ∪{z}的一个最优前缀码。



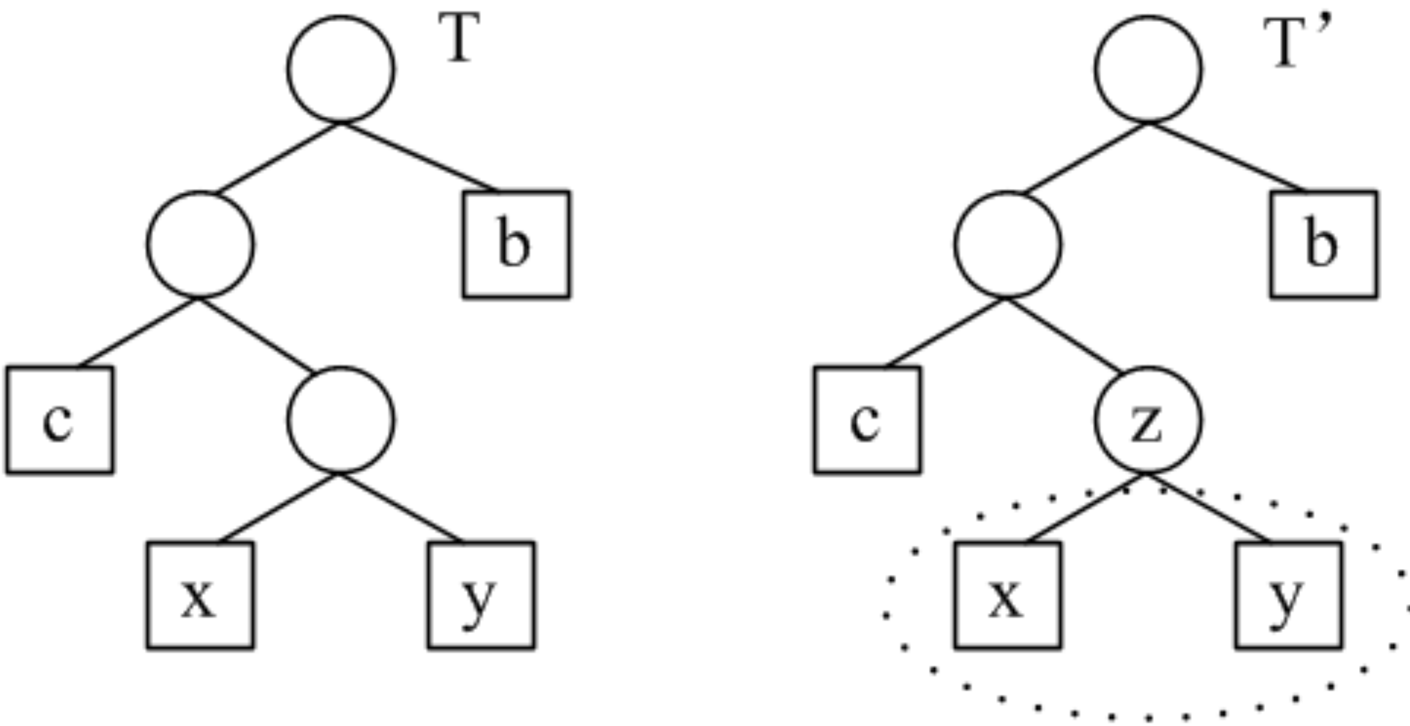
哈夫

证明：

➢ 首先

表示

(T′)



首先，对 c∈ $C-\{x,y\}$，d_T(c)=$d_{T'}$(c)，

故 f(c) d_T(c)=f(c) $d_{T'}$(c)；

其次，d_T(x)=d_T(y)=$d_{T'}$(z)+1，故

f(x) d_T(x)+f(y) d_T(y)=(f(x)+f(y))($d_{T'}$(z)+1)

=f(x)+f(y)+f(z) $d_{T'}$(z)

进而，

$$B(T)=B(T')+f(x)+f(y)$$

最优子结构反证法证明：

- 若 T′ 所表示的字符集 C′ 的前缀码不是最优的，则有 T″ 表示的 C′ 的前缀码使得 B(T″)<B(T′)。由于z被看做是 C′ 中的一个字符，故 z 在 T″ 中是一树叶。若将x和y加入树T″ 中作为z的儿子，则得到表示字符集C的前缀码二叉树 T'''，且有

$$B(T''')=B(T'')+f(x)+f(y)< B(T')+f(x)+f(y)=B(T),$$

这与T的最优性矛盾，故T′ 所表示的C′ 的前缀码是最优的。

4.5 单源点最短路径

单源点最短路径问题描述

给定带权有向图G =(V, E)，其中每条边的权是非负实数。另外，还给定V中的一个顶点，称为源。现在要计算从源到所有其它各顶点的最短路长度。这里路的长度是指路径上各边权之和。这个问题通常称为单源最短路径问题。

Dijkstra （迪杰斯塔拉）算法主要思想

设置顶点集合S并不断地作贪心选择来扩充这个集合。一个顶点属于集合S当且仅当从源到该顶点的最短路径长度已经确定。

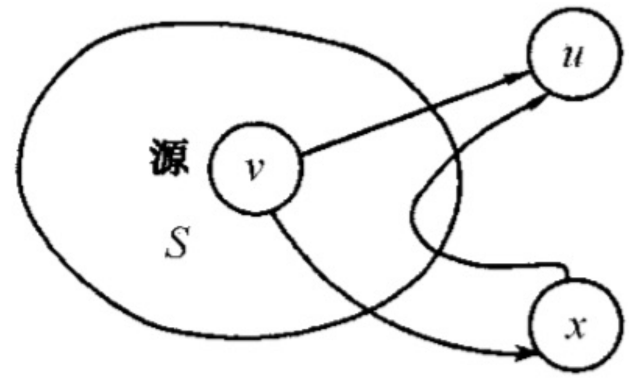
单源点最短路径

算法基本思想（续）：

- 初始时，S中仅含有源。
- 设u是G的某一个顶点，把从源到u且中间只经过S中顶点的路称为从源到u的特殊路径，并用数组dist记录当前每个顶点所对应的最短特殊路径长度。
- Dijkstra算法每次从V-S中取出具有最短特殊路长度的顶点u，将u添加到S中，同时对数组dist作必要的修改。
- 一旦S包含了所有V中顶点，dist就记录了从源到所有其它顶点之间的最短路径长度。

单源点最短路径

- **贪心选择性质**：算法所作的贪心选择是从V-S中选择具有最短特殊路径的顶点u，从而确定从源到u的最短路径长度dist[u]



证明：

- 如果存在一条从源v到u且长度比dist[u]更短的路，设这条路初次走出S之外到达的顶点为x∈ V-S ，然后徘徊于S内外若干次，最后离开S到达u ，如图所示
- 在这条路径上，分别记 d（v ，x），d（x，u）和 d（v，u）为v到x，x到u和v到u的路长，那么

$$\mathrm{dist}[x] \leqslant d(v,x)$$

$$d(v,x)+d(x,u)=d(v,u)<\mathrm{dist}[u]$$

- 因为边权非负，则$d(x,u) \geq 0$，得$dist[x] < dist[u]$，矛盾

□ 最优子结构性质

- 即算法中确定的dist[u]确实是当前从源到顶点u的最短特殊路径长度

证明提示（略）：

- 考察算法在添加u到S中后，dist[u]的值所起的变化。

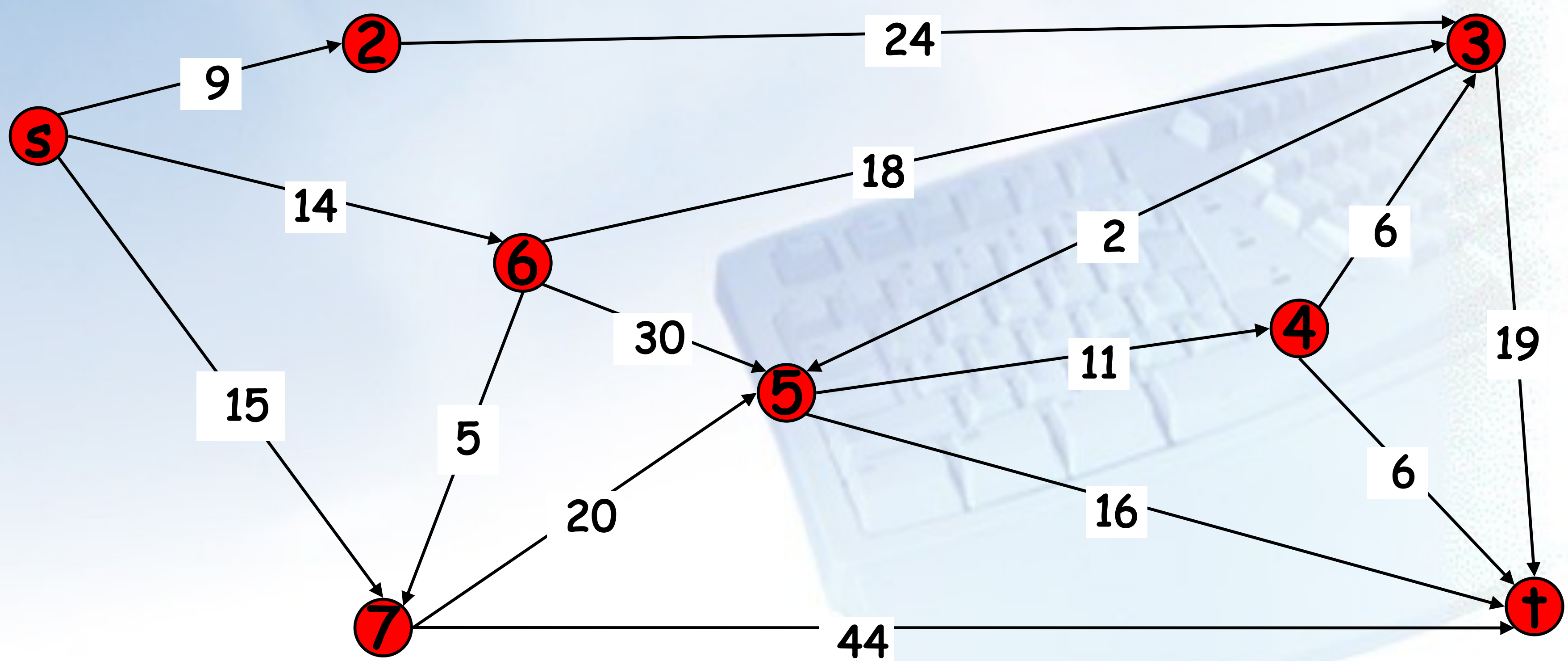
□ 时间复杂性

分析：

- 对于一个具有n个顶点和e条边的带权有向图，如果用带权邻接矩阵表示这个图，那么Dijkstra算法的主循环体需要O（n）时间。
- 这个循环需要执行 n -1次，所以完成循环需要O (n^2)时间。算法的其余部分所需要时间不超过O（n^2）。

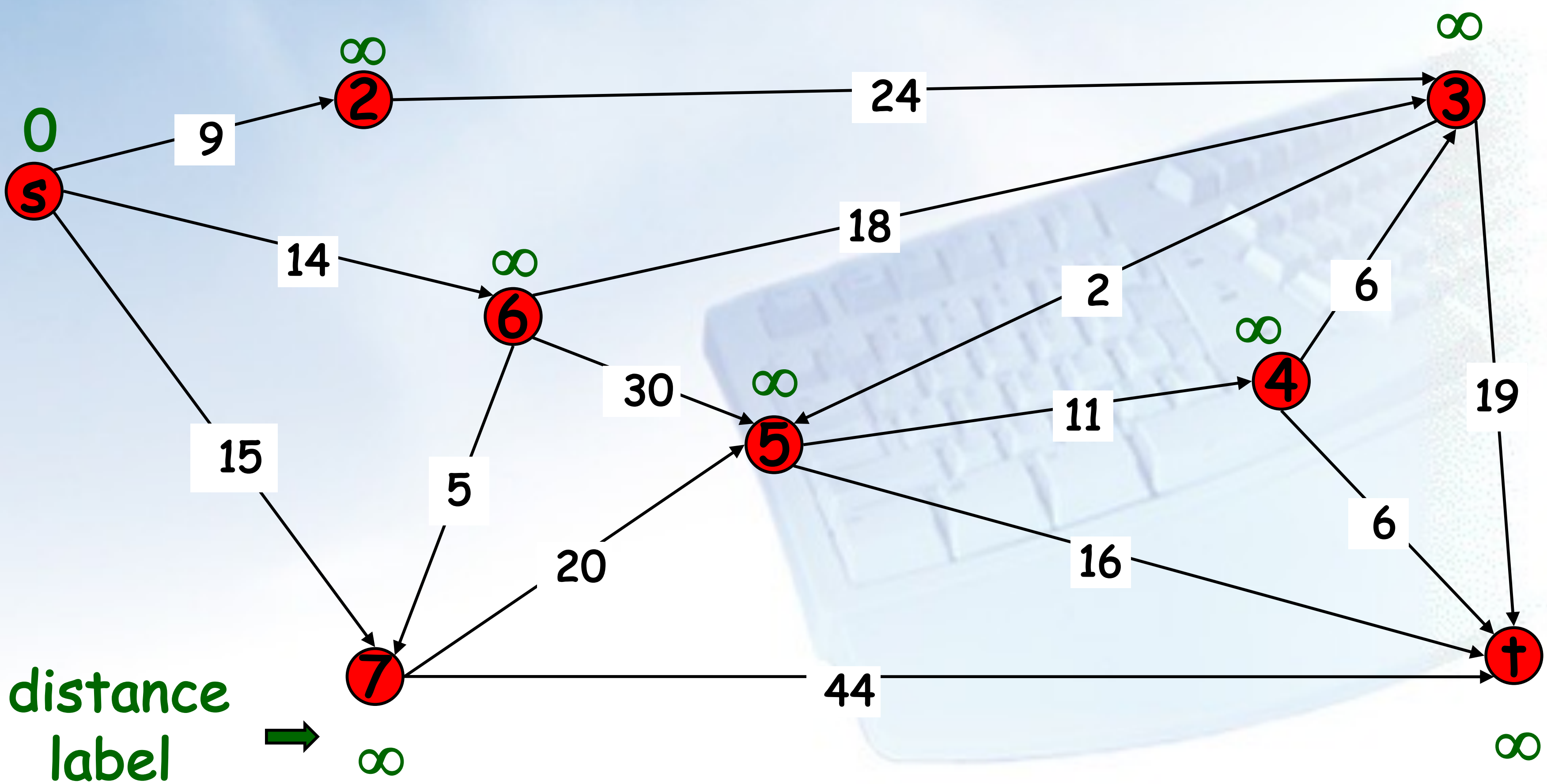
Dijkstra算法示例

- 找到 s 到 t 的最短路径.



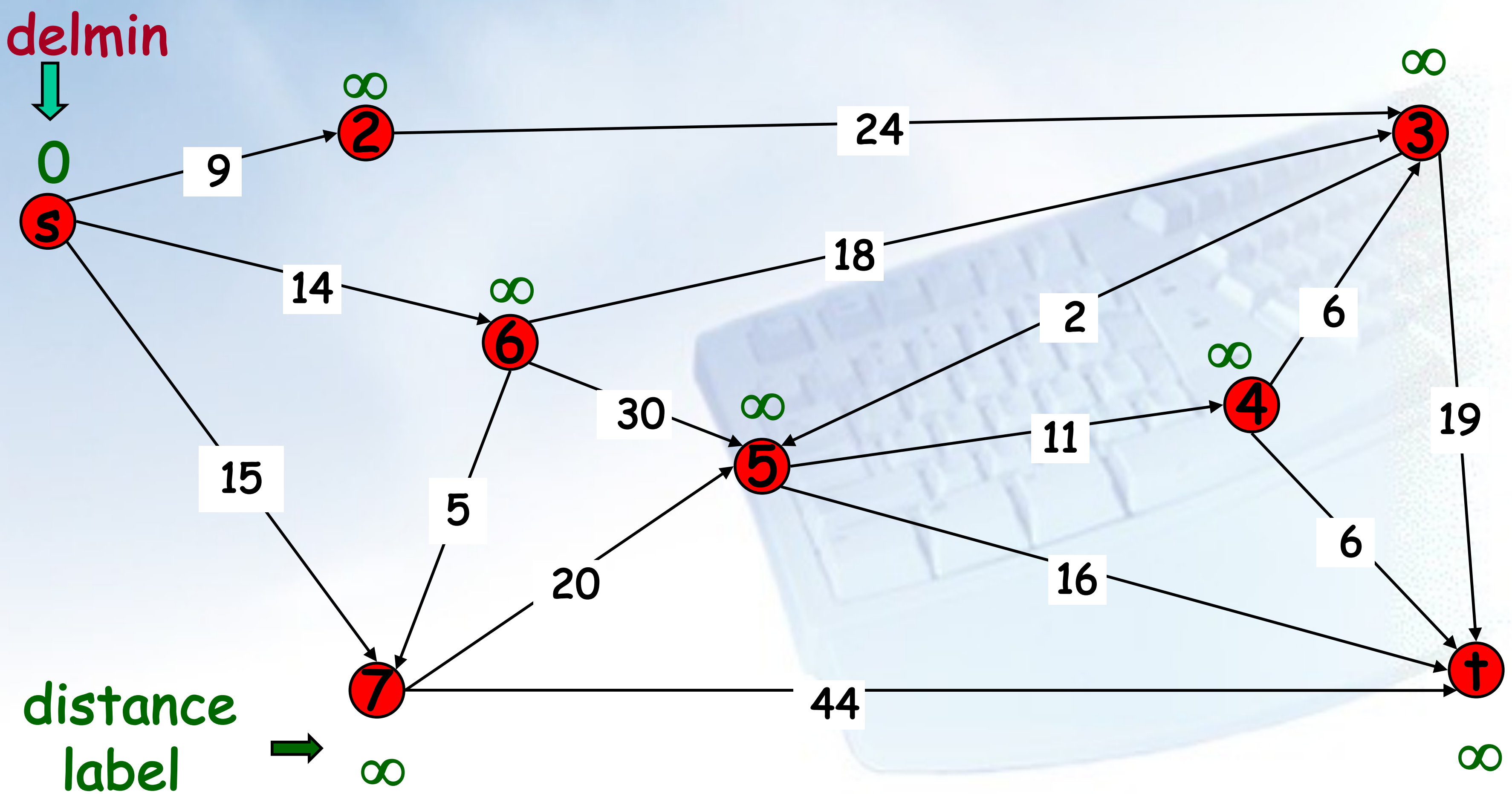
S = { }

PQ = { s, 2, 3, 4, 5, 6, 7, t }



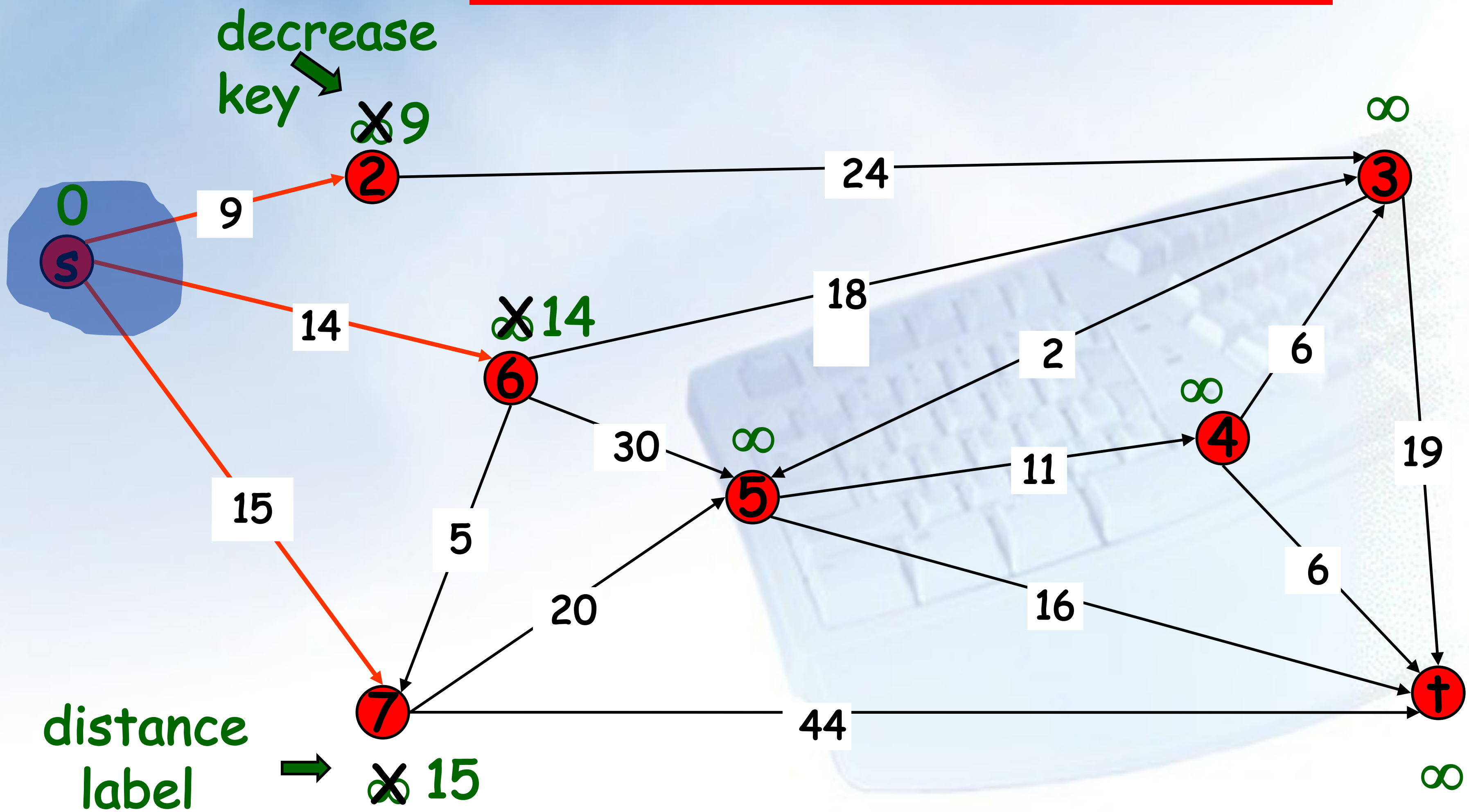
$S = \{\ \}$

$PQ = \{ s, 2, 3, 4, 5, 6, 7, t \}$



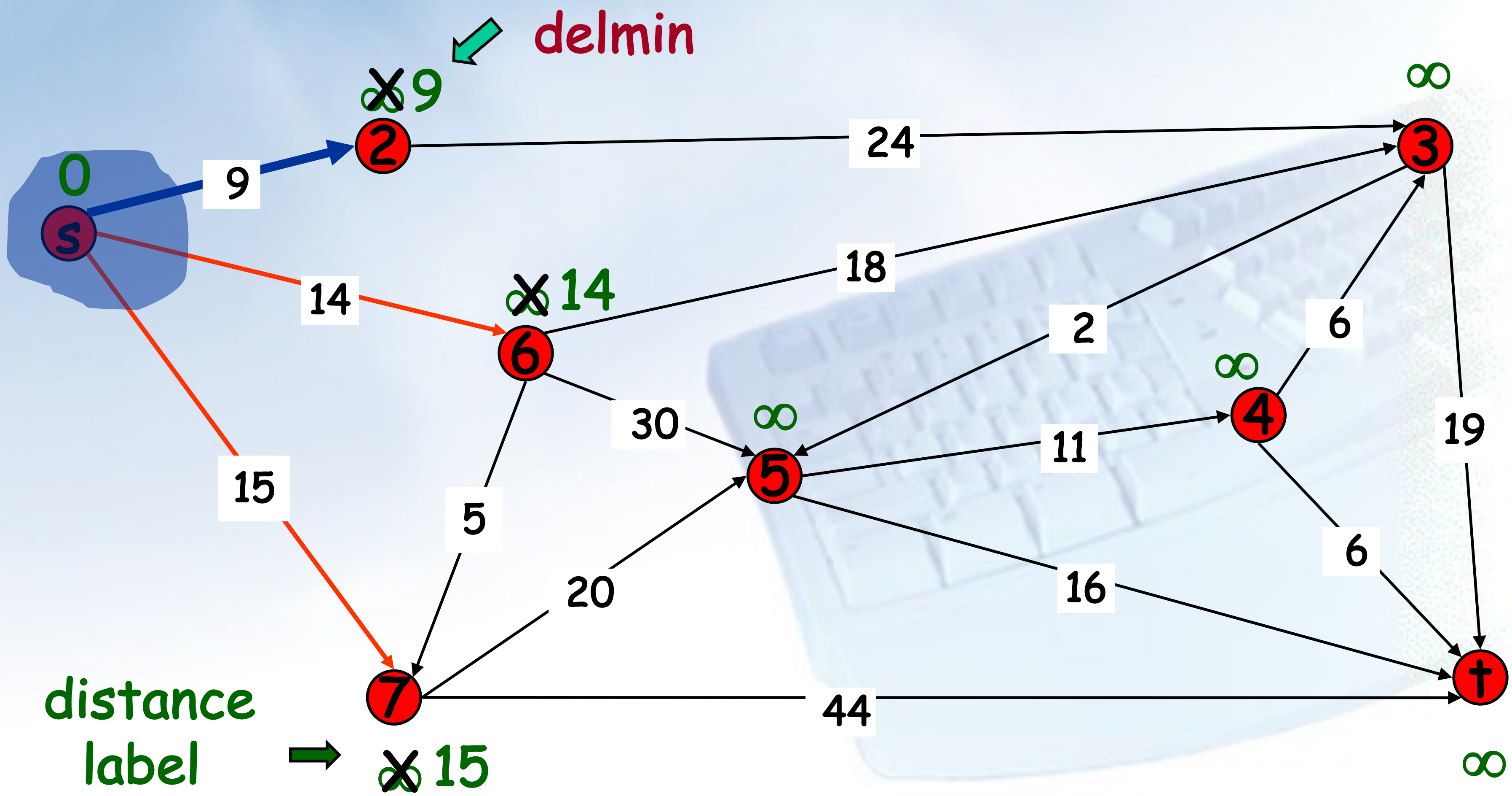
S = { s }

PQ = { 2, 3, 4, 5, 6, 7, t }



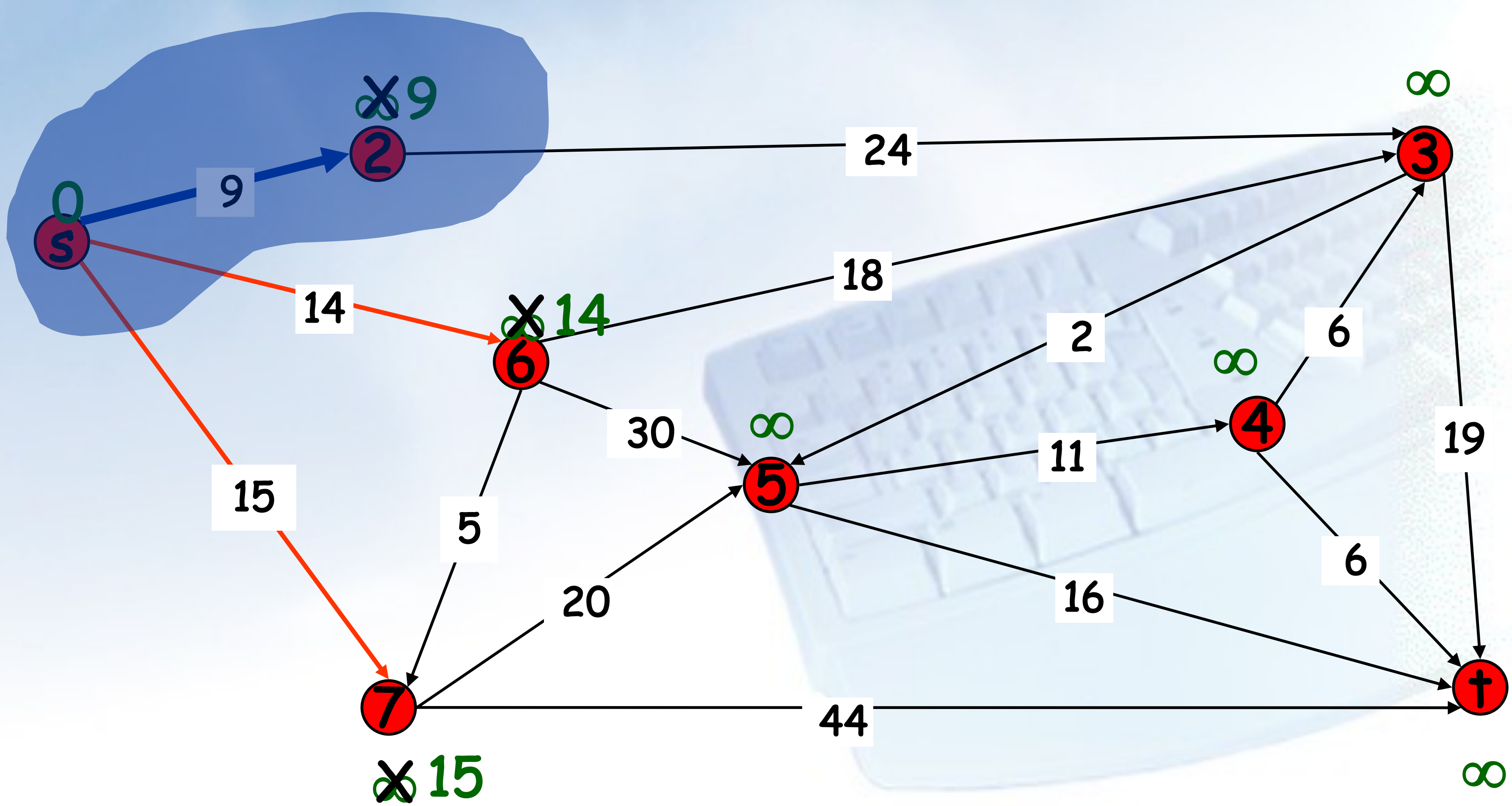
S = { s }

PQ = { 2, 3, 4, 5, 6, 7, t }



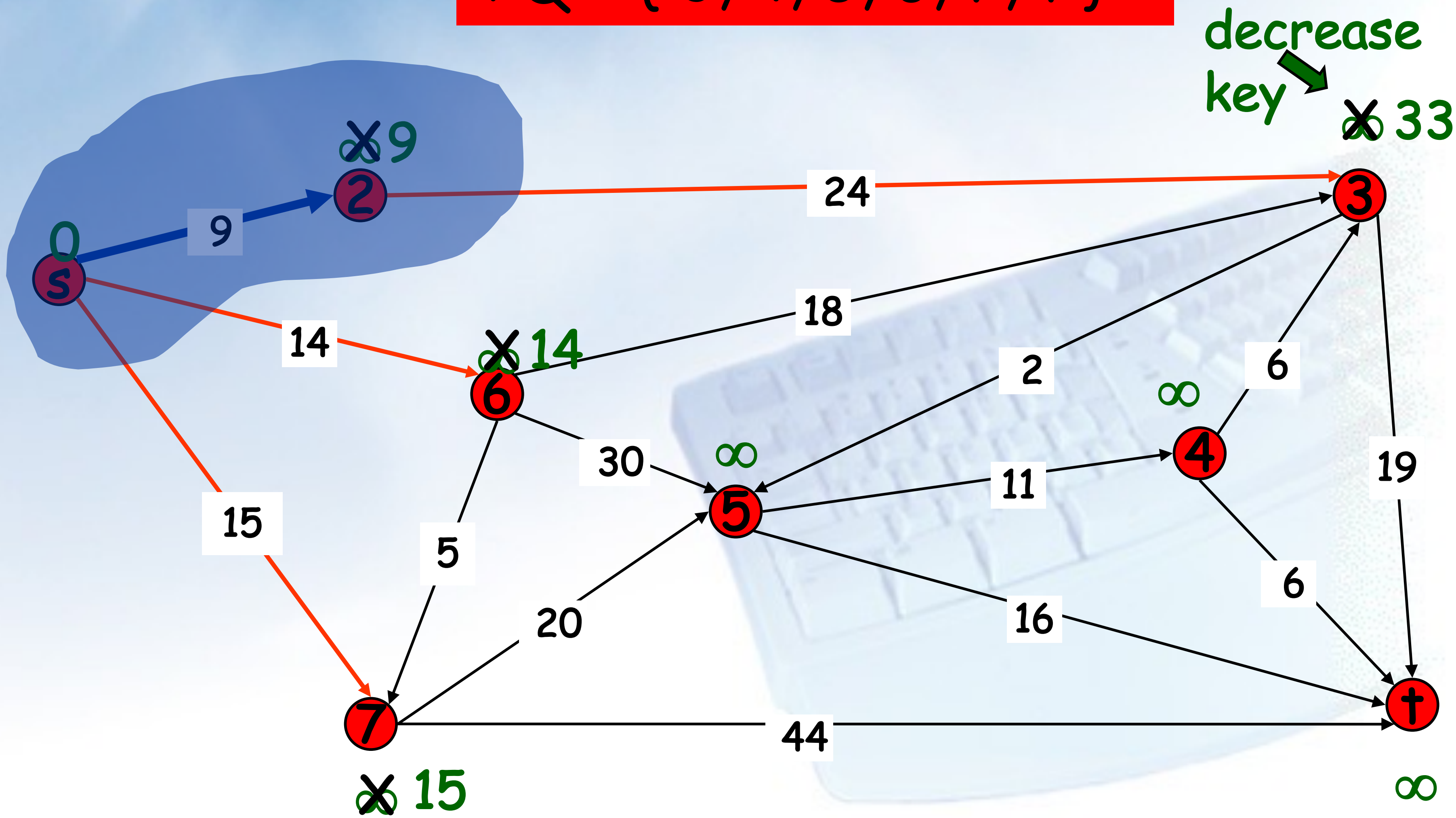
S = { s, 2 }

PQ = { 3, 4, 5, 6, 7, t }



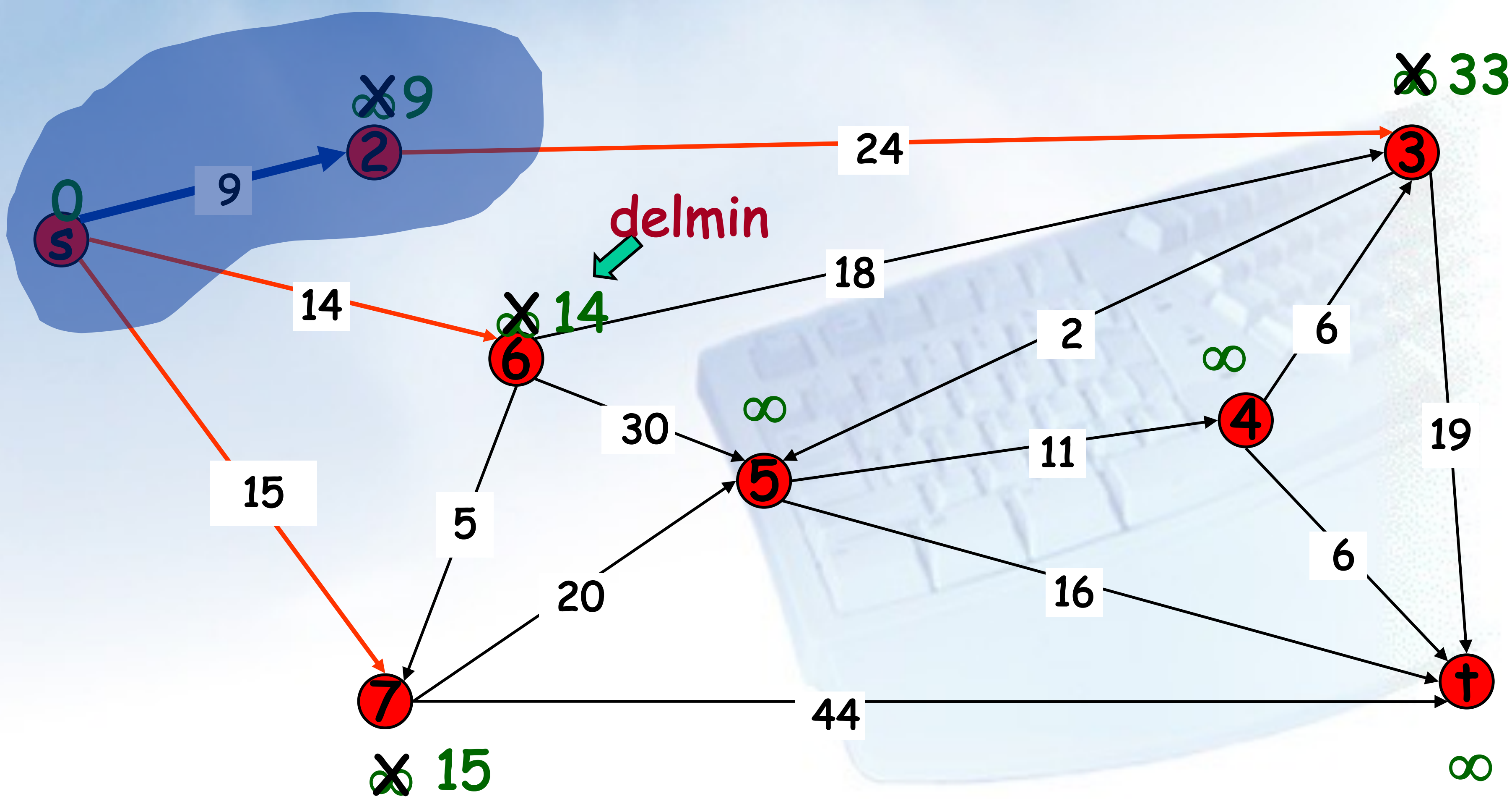
S = { s, 2 }

PQ = { 3, 4, 5, 6, 7, t }



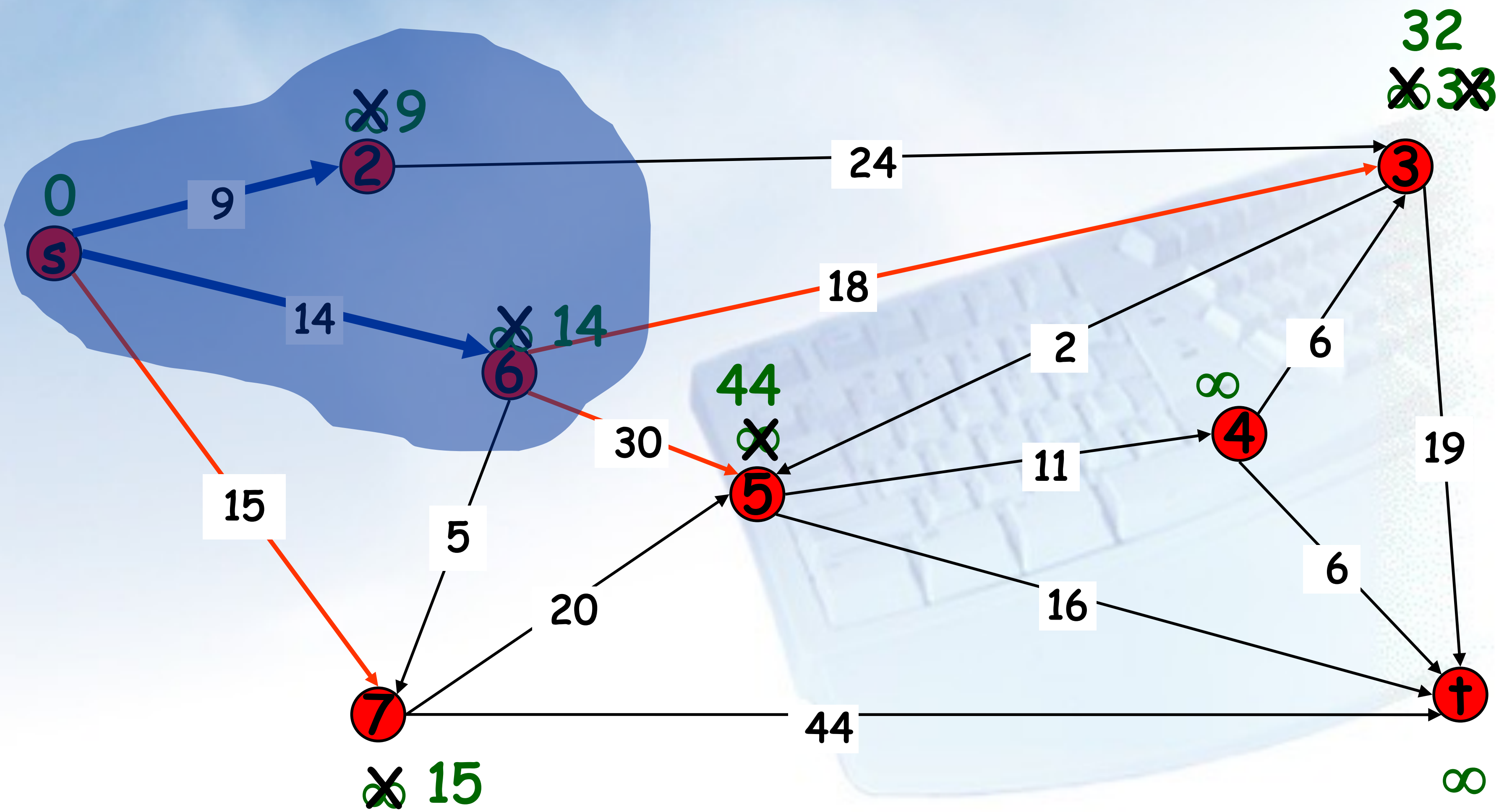
$S = \{ s, 2 \}$

$PQ = \{ 3, 4, 5, 6, 7, t \}$



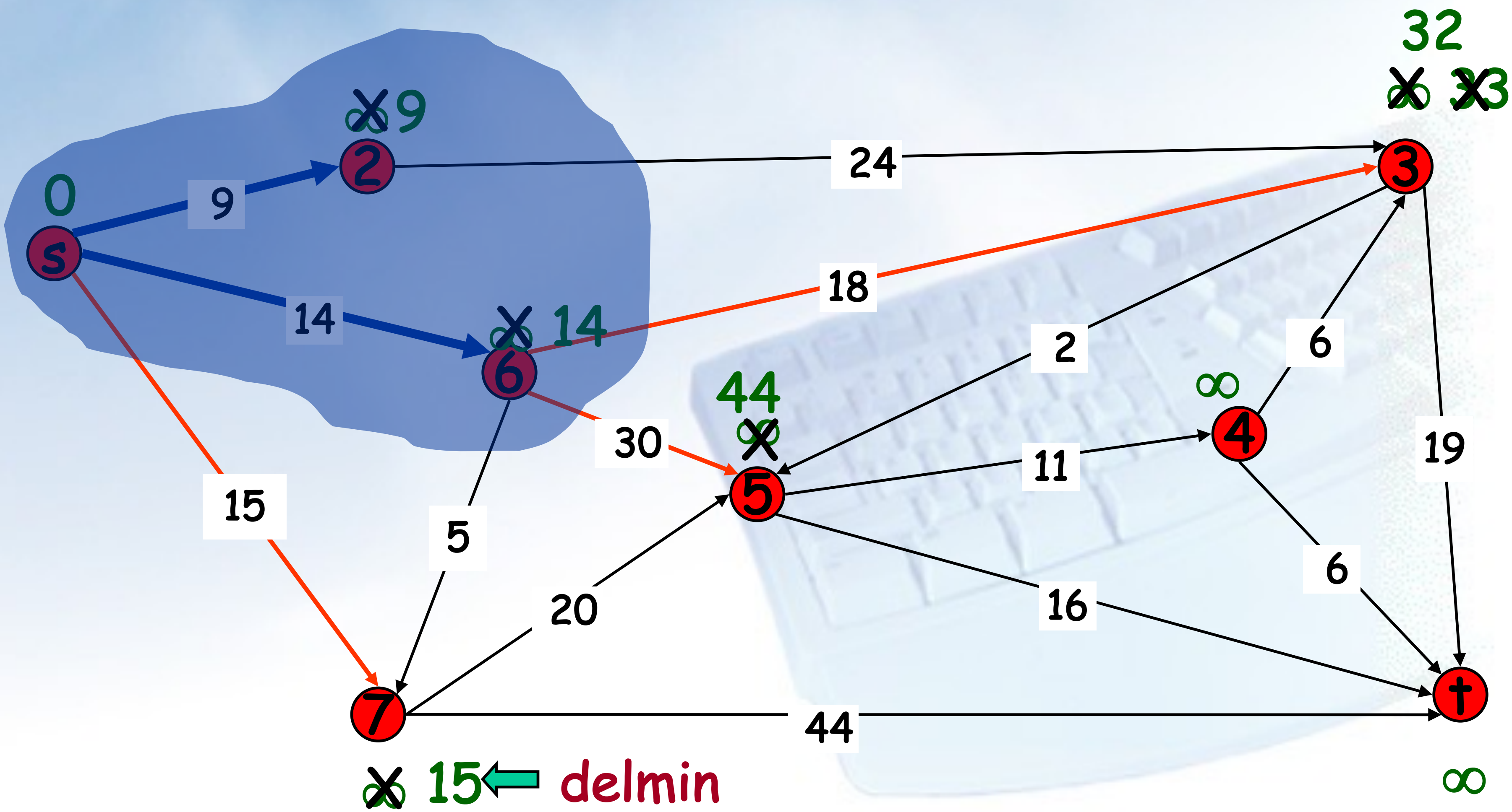
$S = \{ s, 2, 6 \}$

$PQ = \{ 3, 4, 5, 7, t \}$



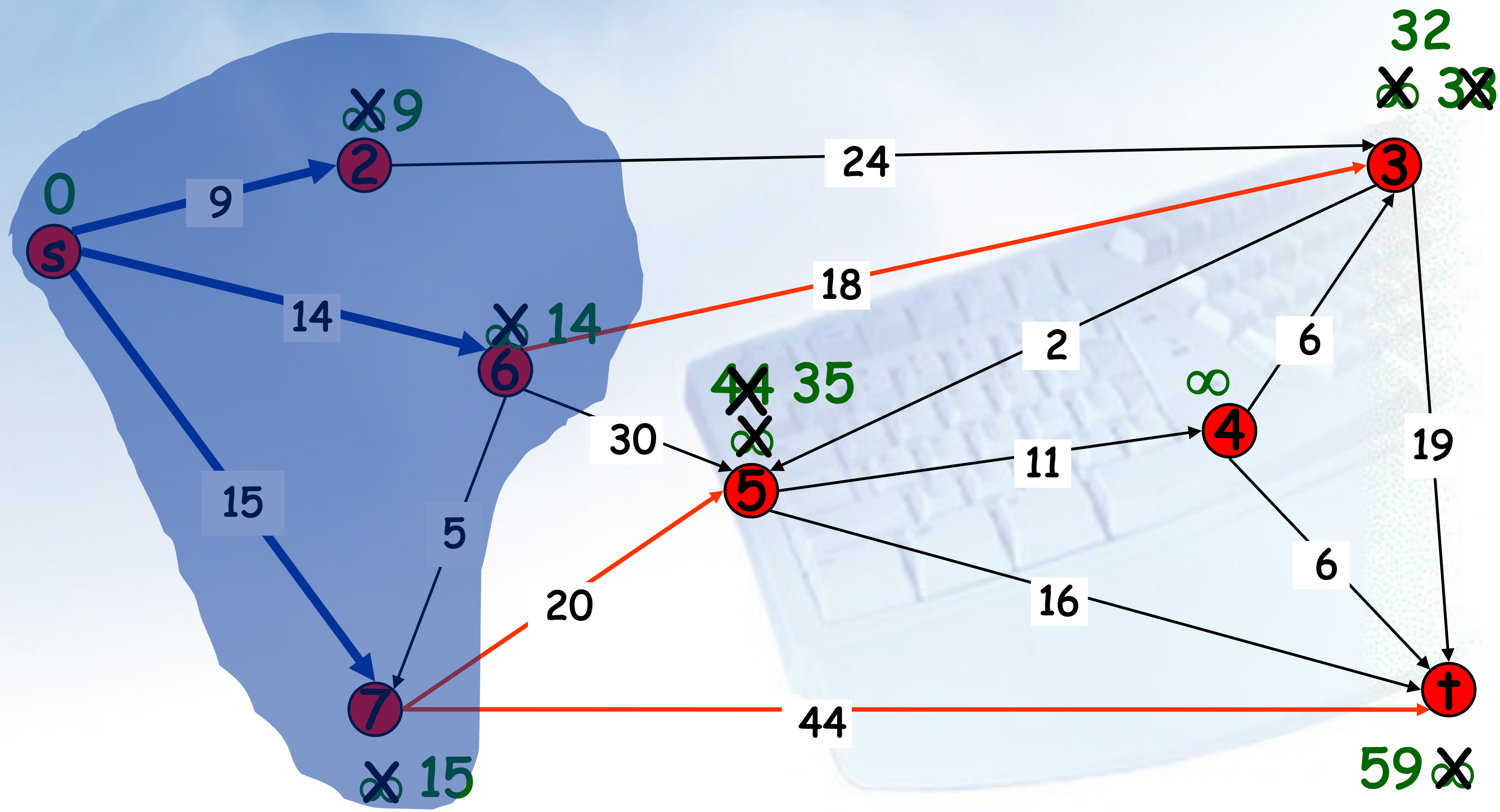
S = { s, 2, 6 }

PQ = { 3, 4, 5, 7, t }



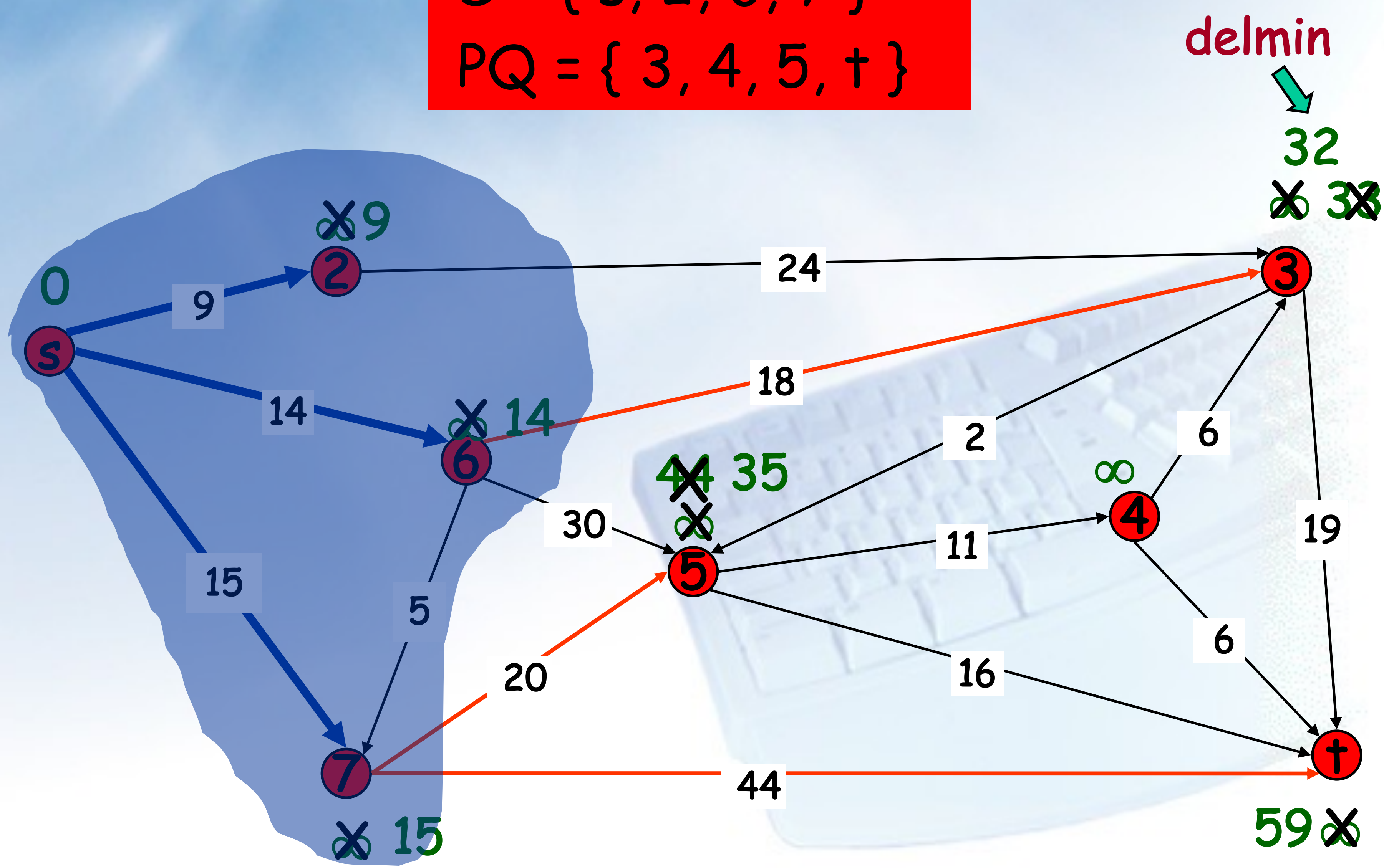
$S = \{ s, 2, 6, 7 \}$

$PQ = \{ 3, 4, 5, t \}$



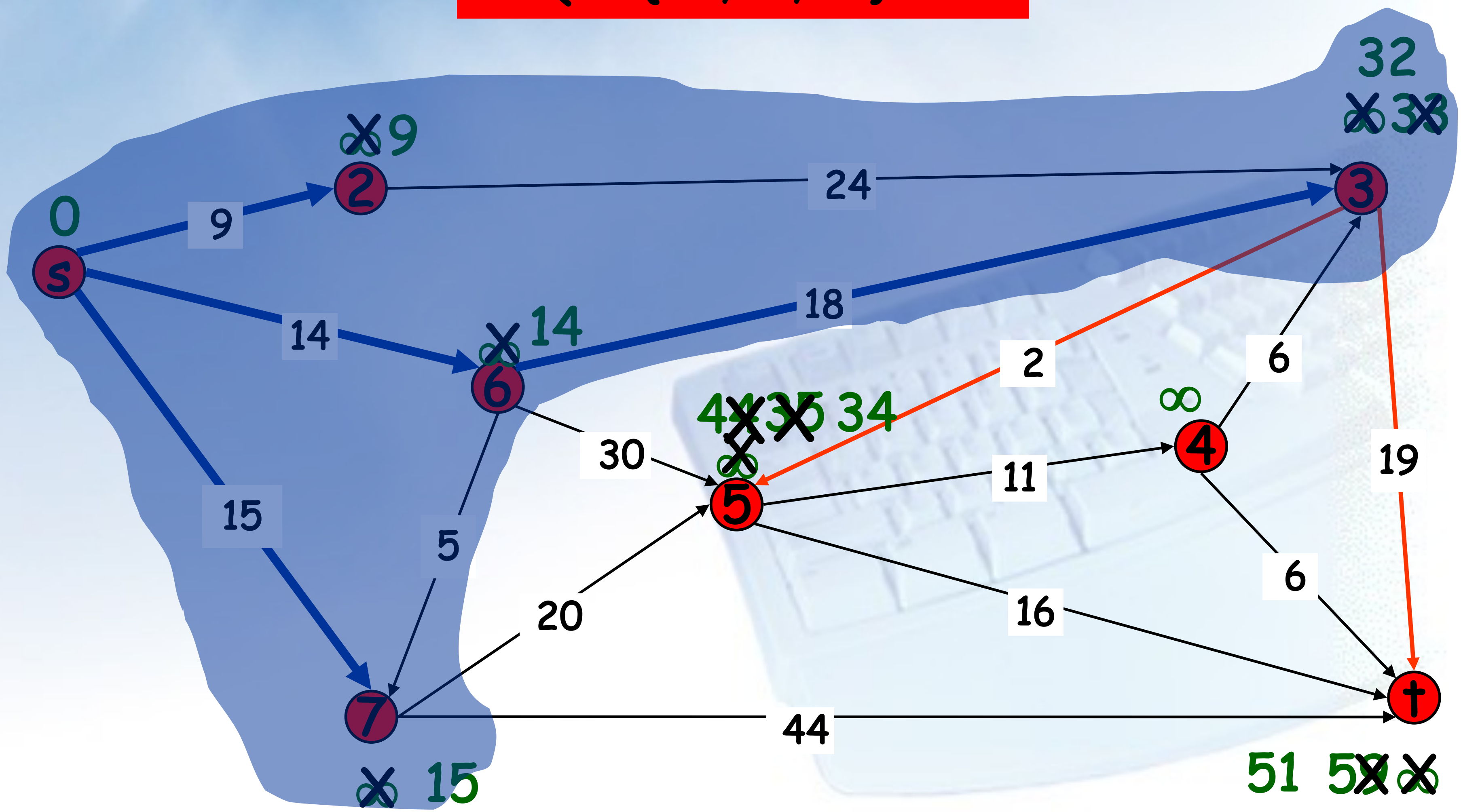
$S = \{ s, 2, 6, 7 \}$

$PQ = \{ 3, 4, 5, t \}$



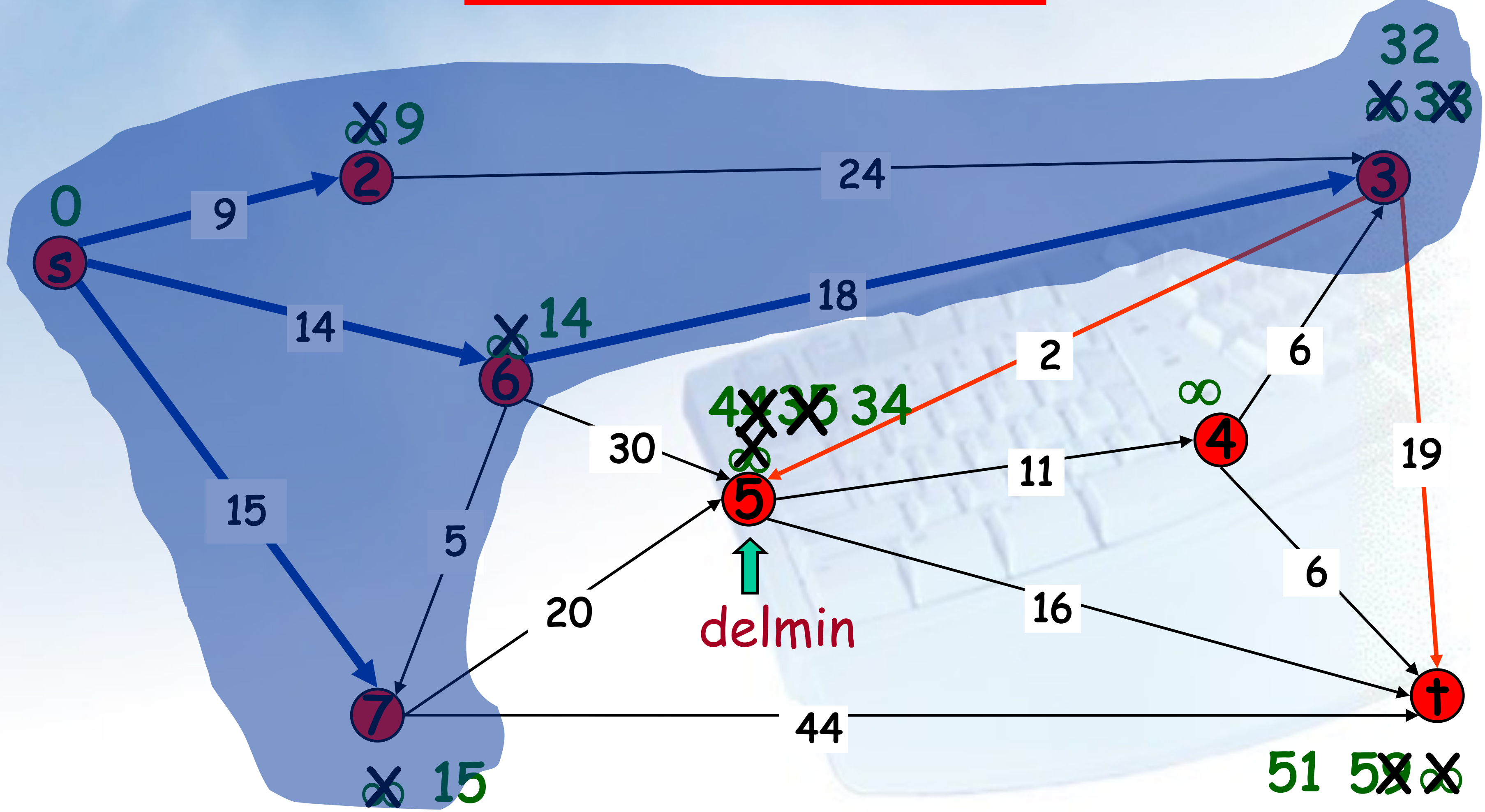
S = { s, 2, 3, 6, 7 }

PQ = { 4, 5, t }



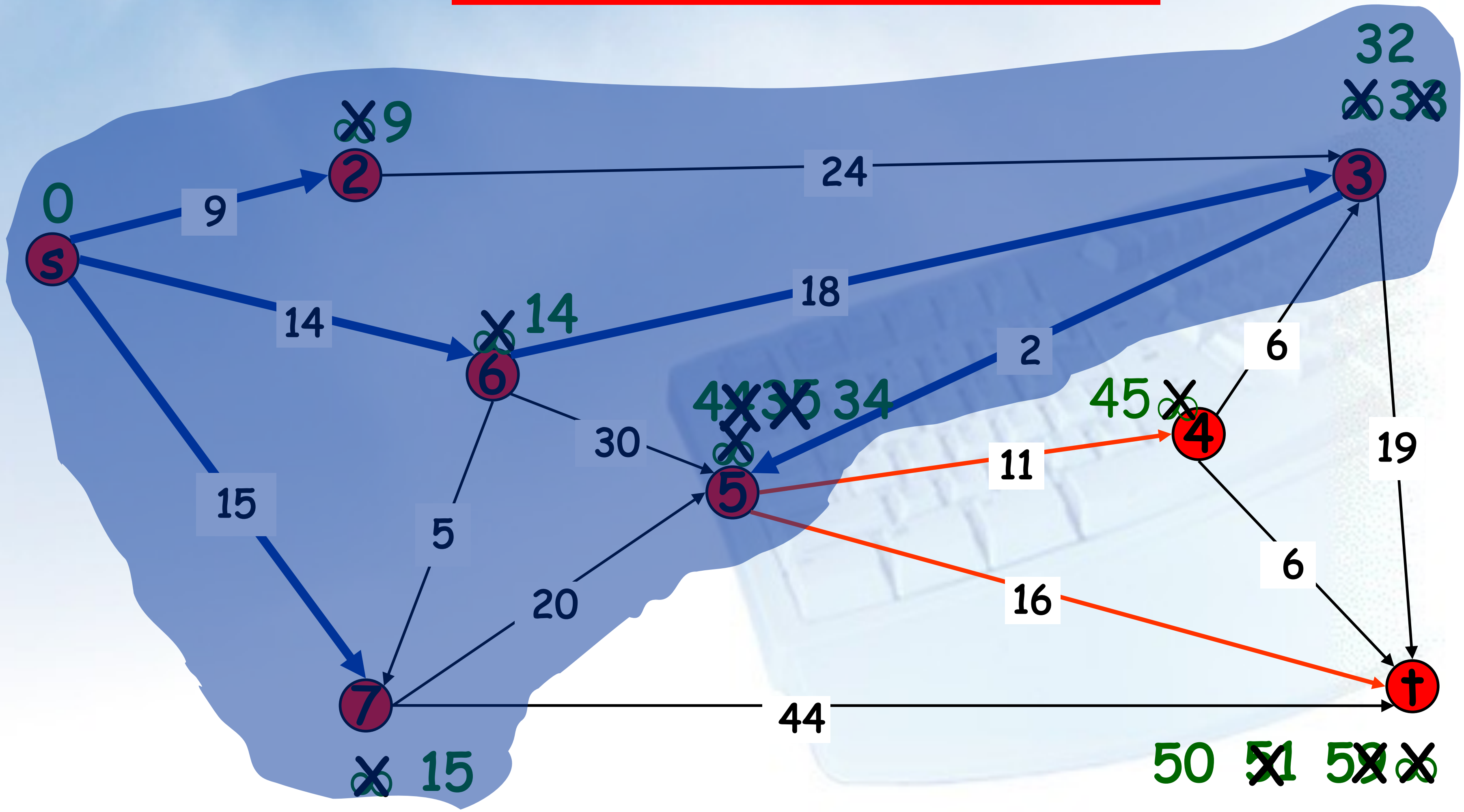
S = { s, 2, 3, 6, 7 }

PQ = { 4, 5, t }



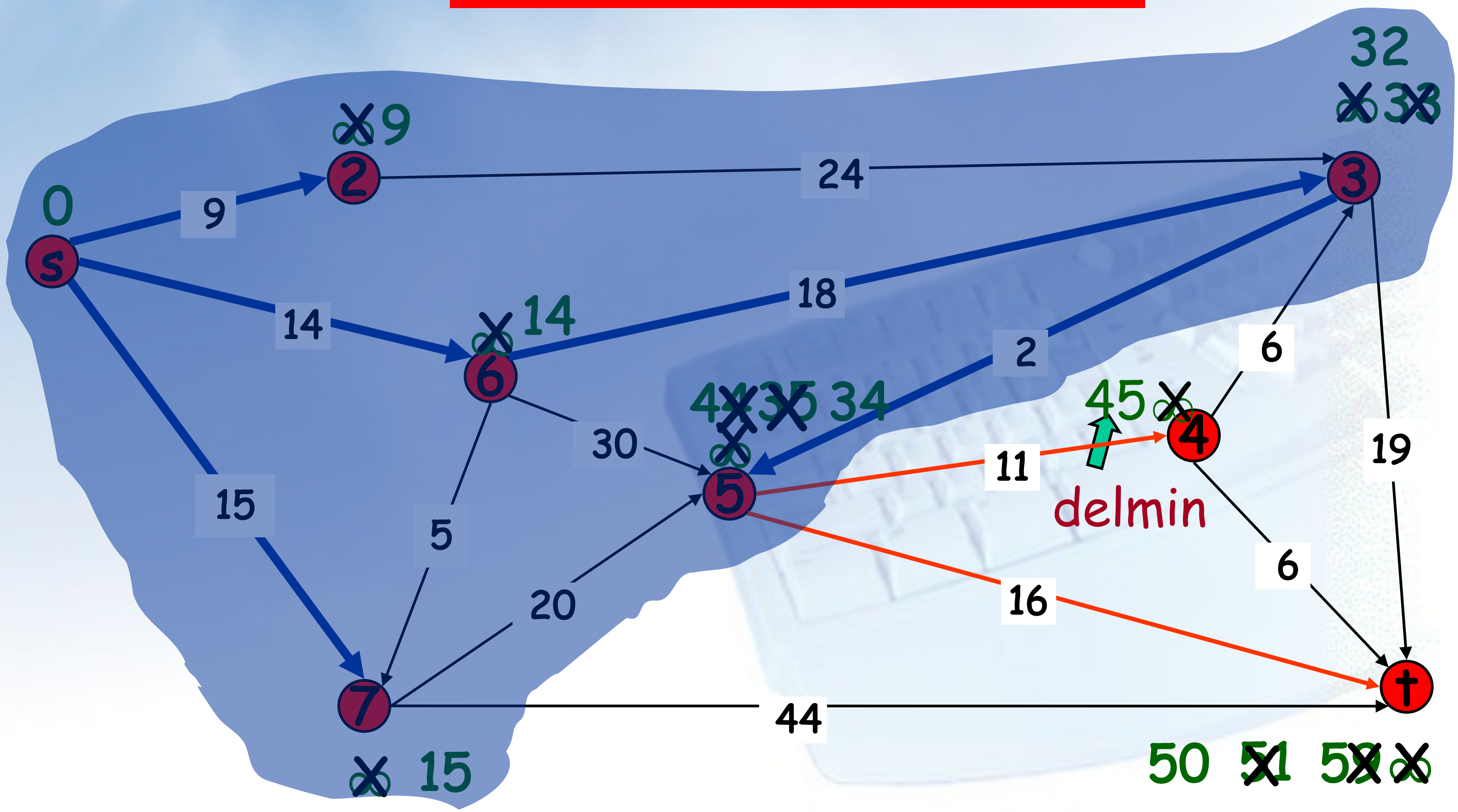
S = { s, 2, 3, 5, 6, 7 }

PQ = { 4, t }



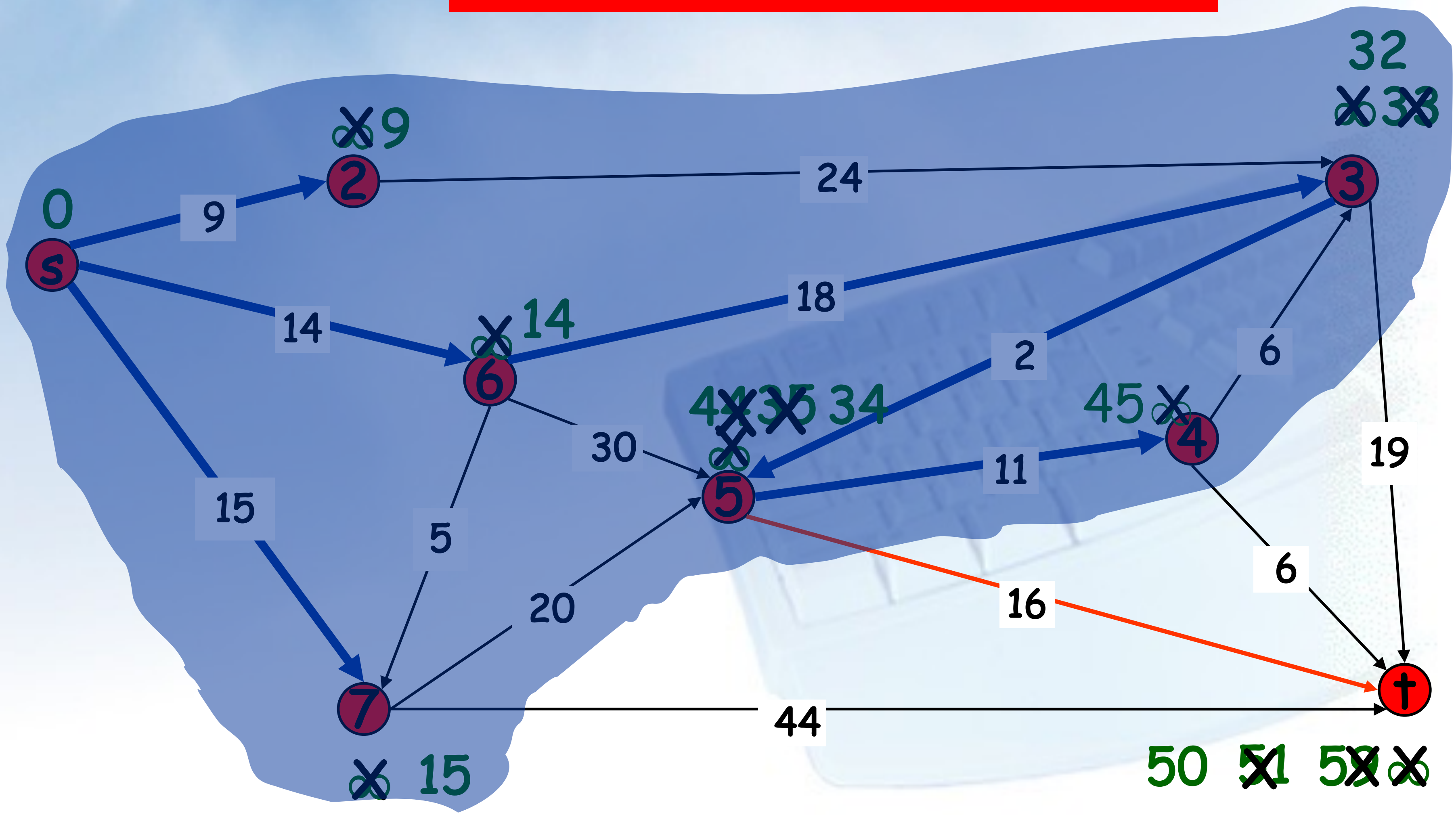
S = { s, 2, 3, 5, 6, 7 }

PQ = { 4, t }



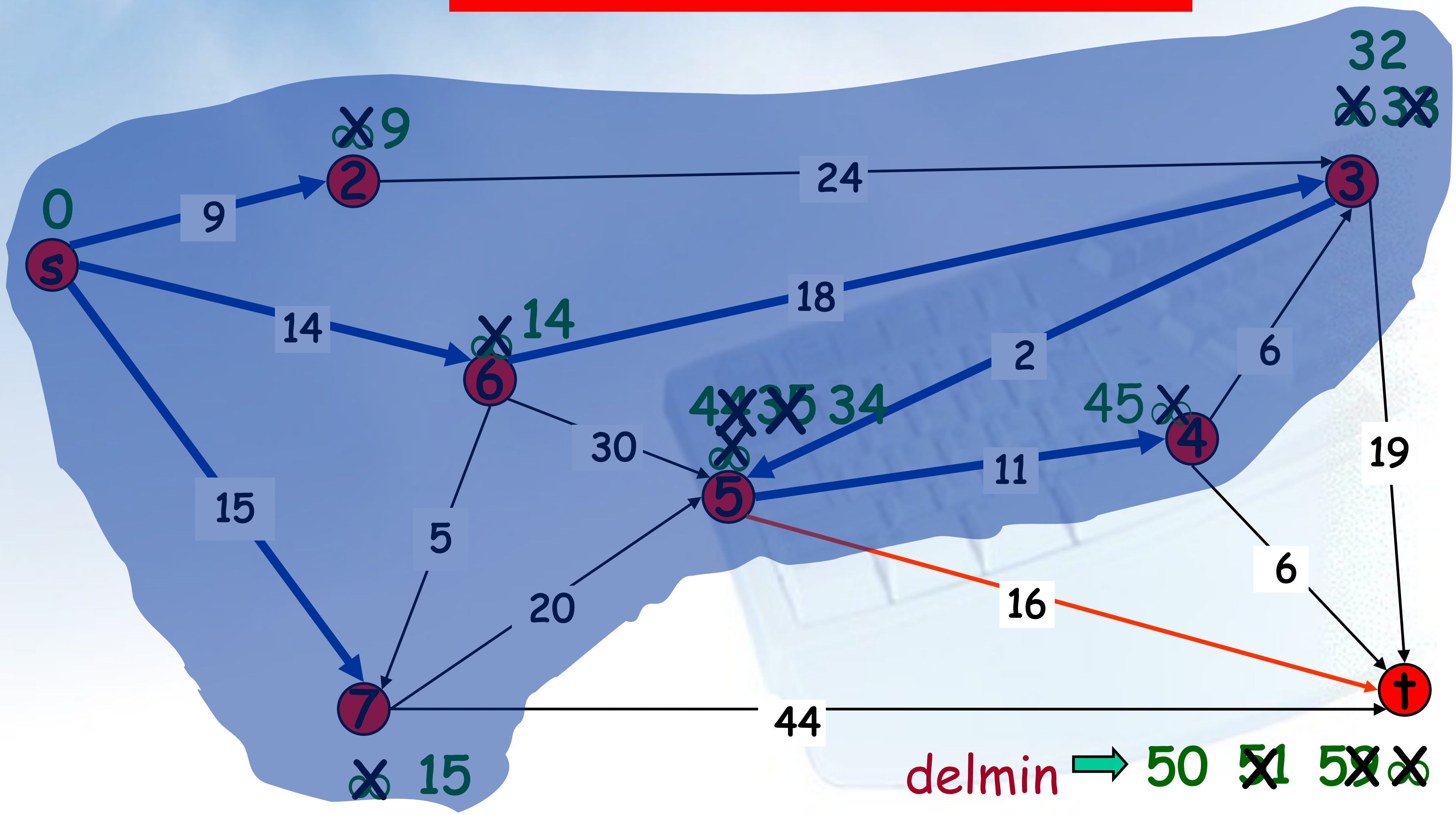
S = { s, 2, 3, 4, 5, 6, 7 }

PQ = { t }

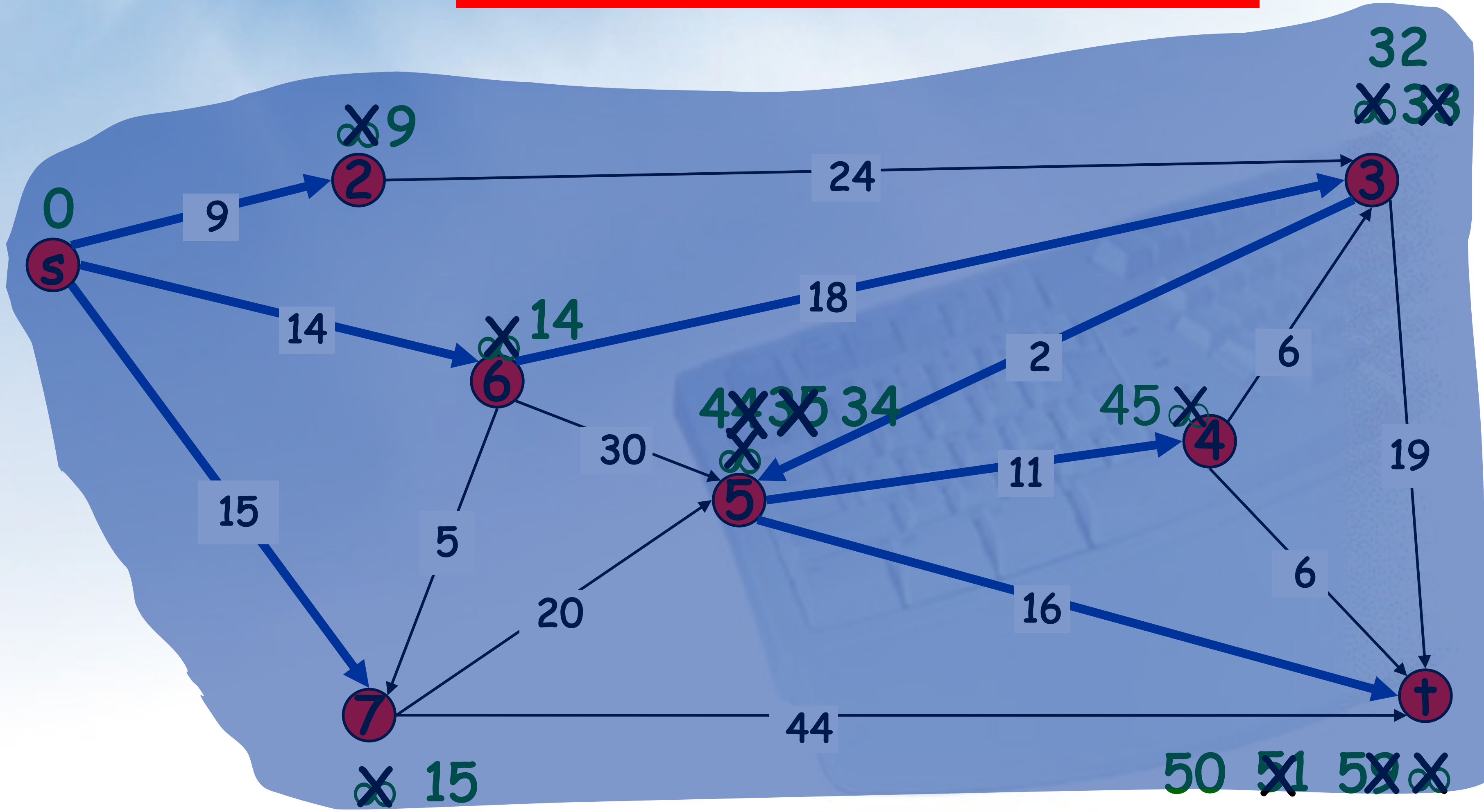


S = { s, 2, 3, 4, 5, 6, 7 }

PQ = { t }

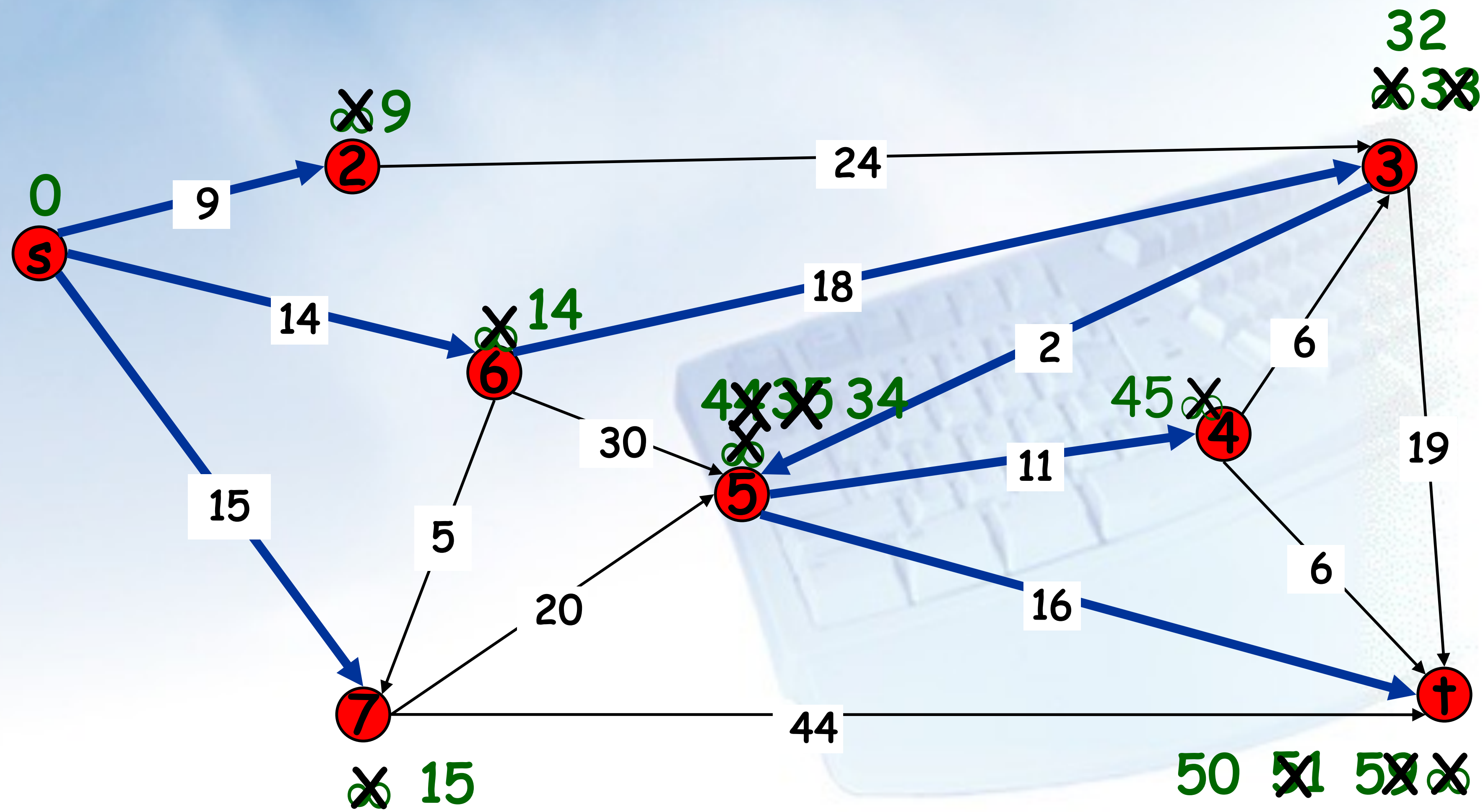


S = { s, 2, 3, 4, 5, 6, 7, t }
PQ = { }



S = { s, 2, 3, 4, 5, 6, 7, t }

PQ = { }



4.6 最小生成树

□ 最小生成树

- 设G =(V,E)是无向连通带权图，即一个网络。E中每条边(v,w)的权为c[v][w]。如果G的子图G'是一棵包含G的所有顶点的树，则称G'为G的生成树。生成树上各边权的总和称为该生成树的耗费。在G的所有生成树中，耗费最小的生成树称为G的最小生成树。
- 网络的最小生成树在实际中有广泛应用。例如，在设计通信网络时，用图的顶点表示城市，用边(v,w)的权c[v][w]表示建立城市v和城市w之间的通信线路所需的费用，则最小生成树就给出了建立通信网络的最经济的方案。

□ 最小生成树性质

- 贪心算法设计策略可以设计出构造最小生成树的有效算
- 本节介绍的构造最小生成树的Prim算法和Kruskal算法都可以看作是应用贪心算法设计策略的例子。尽管这2个算法做贪心选择的方式不同，它们都利用了下面的最小生成树性质：

MST性质：

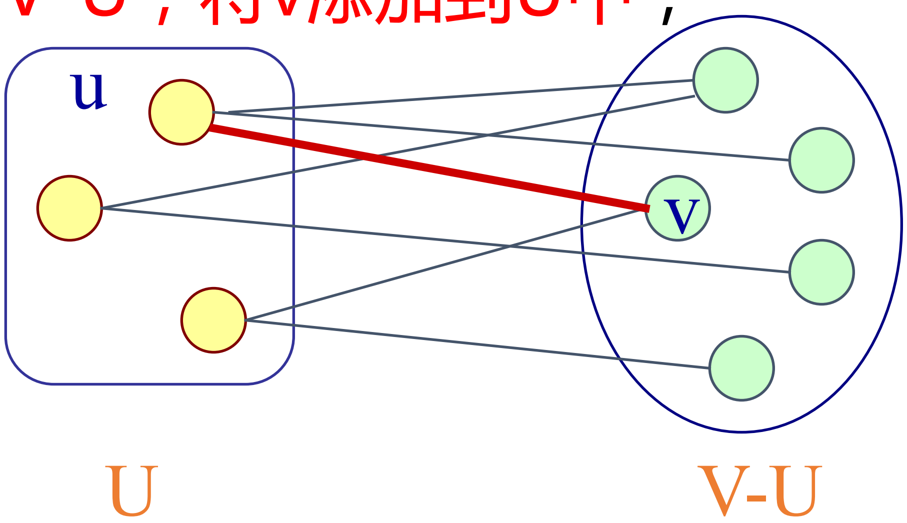
- 设G=(V,E)是连通带权图，U是V的真子集。
- 如果(u,v) ∈ E，且u ∈ U，v ∈ V-U，且在所有这样的边中，(u,v)的权c[u][v]最小，那么一定存在G的一棵最小生成树，它以(u,v)为其中一条边。

普里姆（Prim）算法

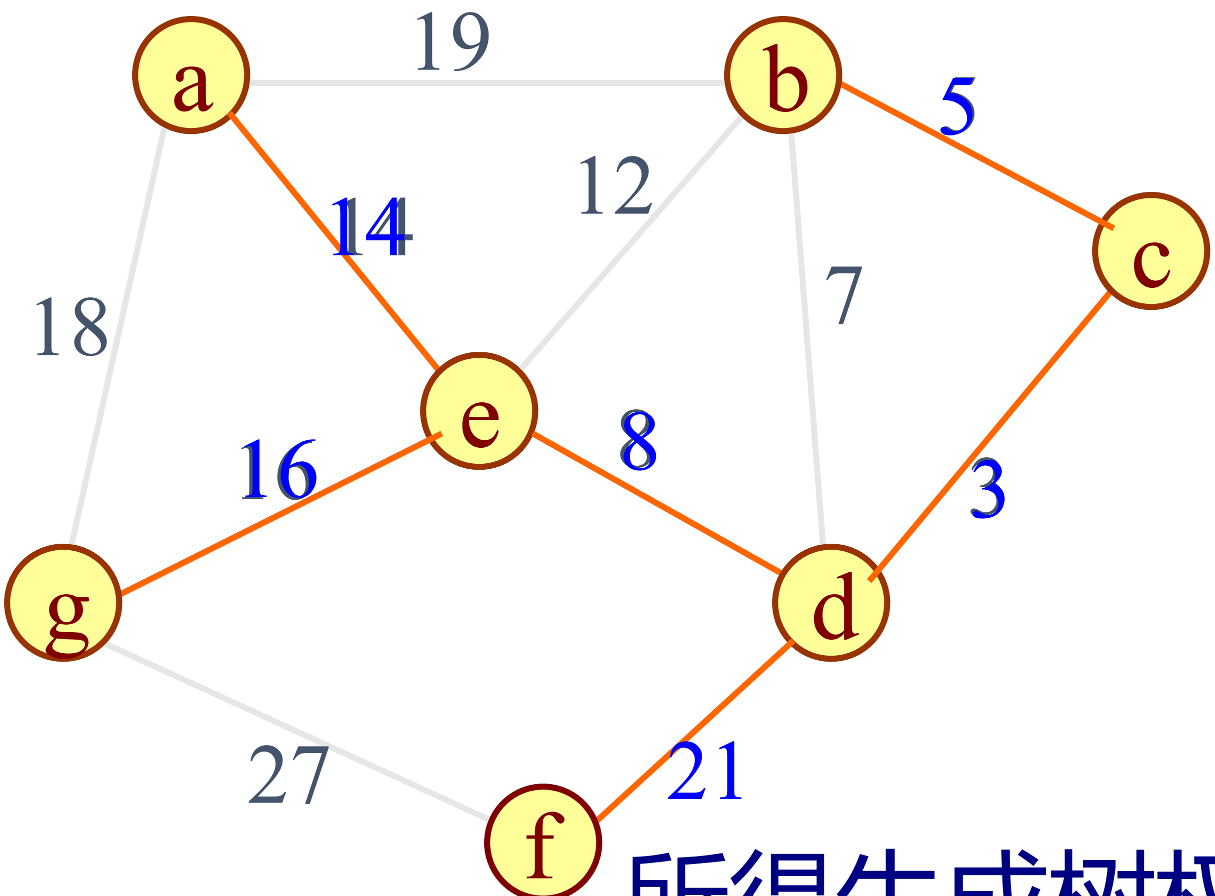
（1）取图中任意一个顶点u作为生成树的根，之后往生成树上添加新的顶点；

（2）在生成树的构造过程中，网中 n 个顶点分属两个集合：已落在生成树上的顶点集 U 和尚未落在生成树上的顶点集V-U ，则应在所有连通U中顶点和V-U中顶点的边中选取权值最小的边(u,v)这里u属于U，v属于V-U，将v添加到U中；

（3）重复（2）继续往生成树上添加顶点，直至生成树上含有 n个顶点为止。



□ 普里姆（Prim）算法示例



所得生成树权值和

$= 14+8+3+5+16+21 = 67$

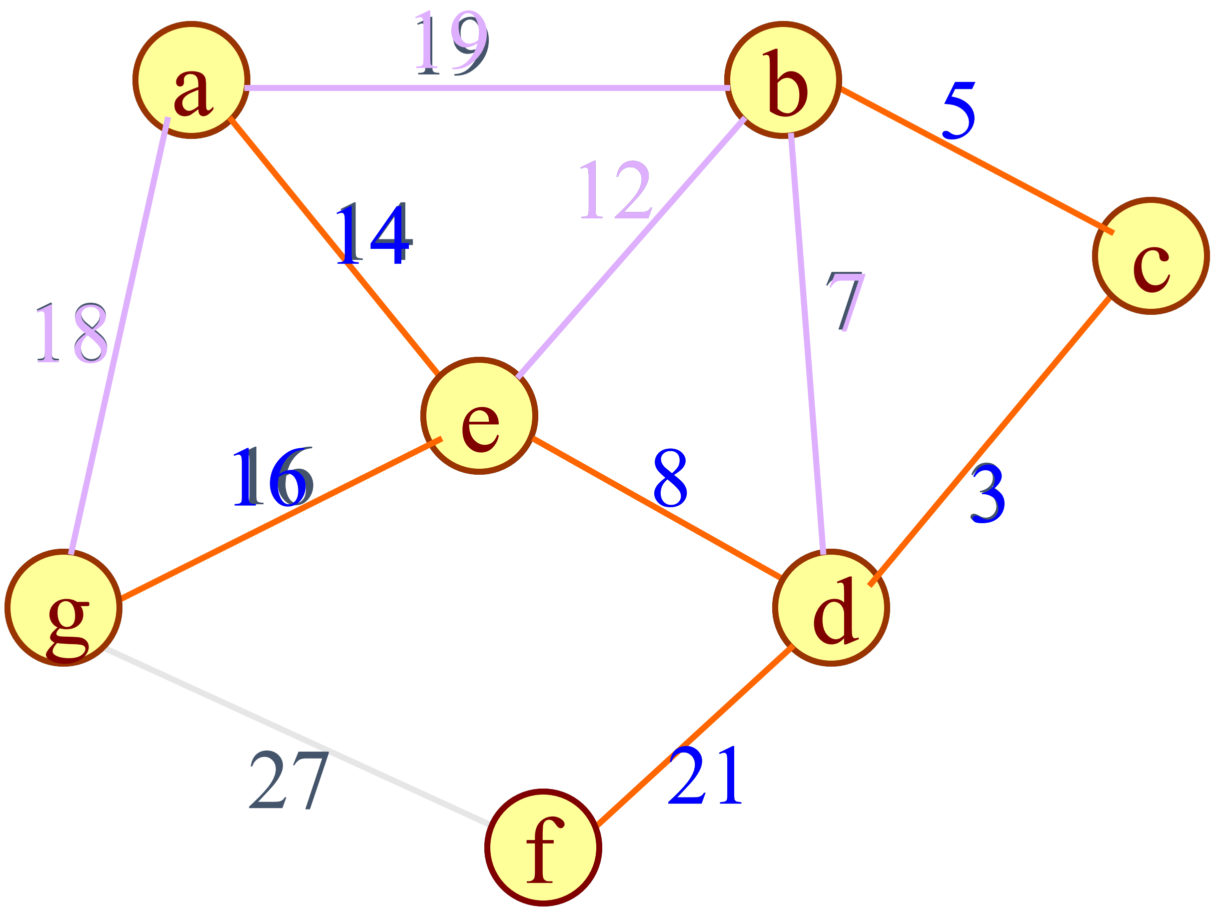
□ 克鲁斯卡尔（Kruskal）算法

为使生成树上边的权值之和达到最小，则应使生成树中每一条边的权值尽可能地小。

□ 具体步骤

- 先构造一个只含 n 个顶点的子图 SG；
- 然后从权值最小的边开始，若它的添加不使SG 中产生长度大于2的简单回路，则在 SG 上加上这条边
- 如此重复，直至加上 n-1 条边为止。

□ 克鲁斯卡尔（Kruskal）算法示例



4.7 多机调度问题

多机调度问题

问题描述

- 多机调度问题要求给出一种作业调度方案，使所给的n个作业在尽可能短的时间内由m台机器加工处理完成
- 这个问题是NP完全问题，到目前为止还没有有效的解法。对于这一类问题,用贪心选择策略有时可以设计出较好的近似算法。

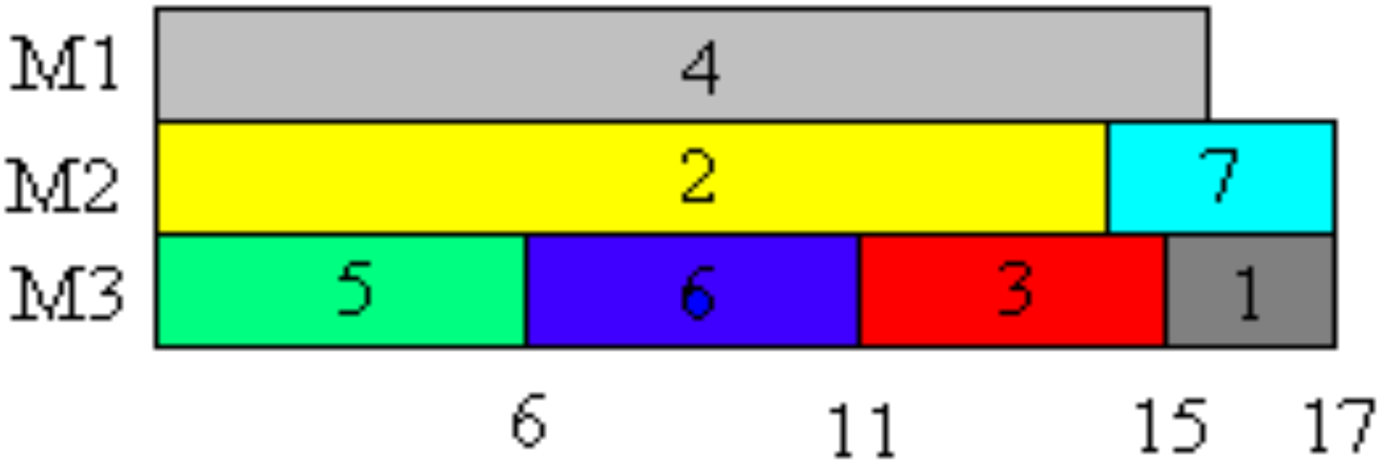
约定：每个作业均可在任何一台机器上加工处理，但未完工前不允许中断处理。作业不能拆分成更小的子作业。

□ 主要思想（n为作业数，m为机器数）

- 贪心选择策略：采用最长处理时间的作业进行优先调度;
- 当n≤m时，只要将机器i的[0,t_i]时间区间分配给作业i即可，算法只需O(1)时间；
- 当n>m时，首先将n个作业依其所需的处理时间从大到小排序，然后依此顺序将作业分配给空闲的处理机，算法所需的计算时间为O(nlogn).

例

- 设7个独立作业{1,2,3,4,5,6,7}由3台机器M1，M2和M3加工处理。
- 各作业所需的处理时间分别为{2,14,4,16,6,5,3}。
- 按算法greedy产生的作业调度如下图所示，所需的加工时间为17。



End