运筹学通论I

胡晓东

应用数学研究所

中国科学院数学与系统科学研究院

http://www.amt.ac.cn/member/huxiaodong/index.html

5. 组合优化-算法设计技巧

精确算法

分而治之 (搜索、排大小序、旅行商)
动态规划 (最短路、三角剖分、背包)
分支定界 (整数线性规划、旅行商、工件排序)

近似算法或启发式算法

贪婪策略 (最小生成树、最大可满足、背包、
顶点覆盖、独立集、旅行商)
局部搜索(最大匹配、旅行商、最大割)
序贯法 (工件排序、装箱、顶点着色)
整数规划法 (顶点覆盖、最大可满足、最大割)
随机方法 (最小割、最大可满足、顶点覆盖)
在线算法 (页面调度、k-服务器、工件排序、装箱)
不可近似 (最大团、背包、旅行商、装箱、连通控制集等)

5.2 动态规划

首先回忆一下，**分而治之方法**是将一个问题分解为若干个相互独立的子问题，然后用递归的方式求解子问题，最后将子问题的解结合起来从而得到原问题的解。

与该方法不同的是，**动态规划方法**是将一个问题分解为若干个彼此非独立的子问题，亦即，几个子问题之间分享若干个子子问题。应用动态规划方法求解一个优化问题通常包括如下几个步骤：

- 刻画最优解的结构，
- 给出最优值的递归表达式，
- 计算最优值，
- 根据所得最优值，确定最优解。

5.2 动态规划（续一）

可以看出，由于采用分而治之方法设计的算法重复求解了相同的子子问题，所以其所作的一些运算是不必要的。而采用动态规划方法设计的算法求解每一个子子问题仅一次，然后将所得解存贮在一个表中，这样就避免了遇到同一个子子问题的时候，需要再花时间求出它的解。

粗略地讲，动态规划方法可以用于求解这样的问题：其最优解可以由有限多个子问题的最优解构成，（不必考虑子问题的解是如何得到的），通常称其为**最优性原则**。

需要强调的是，当我们描述一个动态规划算法的时候，是用**自上而下**的方式；而当我们运行该算法时，是用**自下而上**的方式。

5.2 最短路

首先，我们考虑著名的求两点之间的**最短路问题**。

问　题：最短路

实　例：一个连通图 $G(V, E)$，其中每条边 $(u, v) \in E$ 赋值一个权重 $w(u, v)$，一对儿顶点 s 和 t。

可行解：图 G 中一条由 顶点 s 到 顶点 t 的路 $P(s, t)$。

目　标：极小化路 $P(s, t)$ 的长度，$w(P) = \sum_{(u, v) \in P} w(u, v)$

注意，每条边的权重不仅可以表示两端点之间的距离，还可以表示时间、费用等，只要一条路的权重等于该路上每条边的权重的（线性）求和，而且我们希望这个权重最小。

5.2 最短路（续一）

我们首先假设每一条边的权重是一个非负实数。如果所有的权重都是一样的，那么最短路问题可以在线性时间内求解**(练习)**。如果所有的权重都是自然数，那么我们可以将此情形转化为权重都一样的情形，然后求解。然而，这不再是一个线性时间算法了，因为在转化过程中，我们可能引入了输入的指数多条新边。

一个非常有效的算法就是**Dijkstra算法**（1959年），它采用的就是**动态规划**方法。它利用了最短路的一个简单性质：两点之间的一条最短路包含有若干条最短路。这个**最优子结构**性质是我们采用**动态规划**方法和**贪婪策略**的一个本质特征。

5.2 最短路（续二）

下述引理给出了最短路的**最优子结构**性质的刻画。为了叙述方便和简单，我们用 $p(s, t)$ 表示由顶点 s 到顶点 t 的一条最短路，同时还用它表示其长度。

引理 1 给定一个赋权有向图 $G=(V, E)$ 及两个顶点 $s, t \in V$。设 $p(s, t) = v_1 v_2 \ldots v_k$ 是图G中由 $s = v_1$ 到 $t = v_k$ 的一条最短路，另对任意指标 i 和 j，$1 \le i \le j \le k$，设 p_{ij} 为路 $p(s, t)$ 上由顶点 v_i 到顶点 v_j 的子路。则 p_{ij} 是图G 中一条由 v_i 到 v_j 的最短路。

引理 2 设 $G=(V, E)$ 是一个赋权有向图，且 $s \in V$。则对于每一条边 $(u, v) \in E$，都有 $p(s, v) \le p(s, u) + w(u, v)$。

练习：分别证明**引理1** 和**引理2**。

5.2 最短路（续三）

由上述引理可以导出以下递归关系式：

$$p(s, s) = 0,$$

$$p(s, u) = \min\{ p(s, v) + w(v, u) \mid (v, u) \in E(G)\}。$$

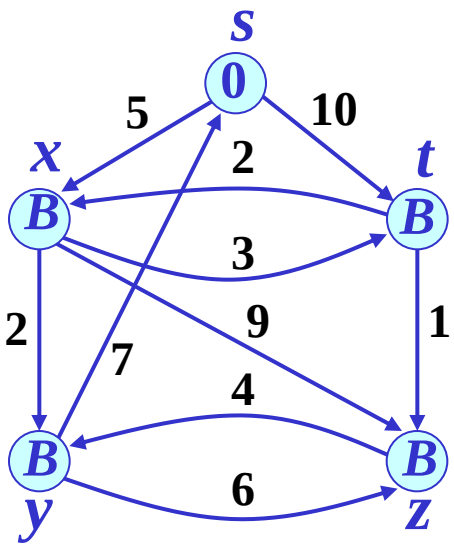
我们下面来讲述 **Dijkstra算法**。用顶点集合 S 表示由顶点 s 到它的最终最短路已经确定。最初，S 仅包含顶点 s。然后，算法重复选取一个顶点 $u \in V \setminus S$，估计其最短路长，并将 u 放入集合 S 中直到 S 包含顶点 t。 在运行算法时，我们用 $p[v]$ 表示从 s 到 v 的最短路的上界。最初，对每一个顶点 $v \notin V \setminus \{s\}$，我们取 $p[v]$ 都为 ∞；当算法终止时， $p[v]$ 即为从 s 到 v 的最短路长，即 $p[v] = p(s, v)$。在算法运行中， 我们用 $p(v)$ 表示从 s 到 v 的最短路上到达 v 的前一个顶点；这样当算法终止时，我们不仅可以得到最短路长，而且可以确定相应的最短路。

5.2 最短路（续四）

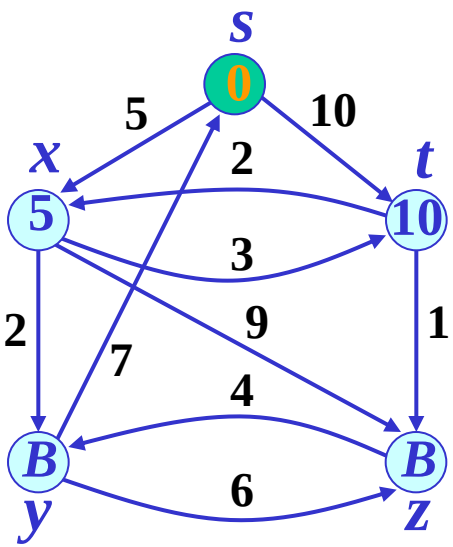
ALGORITHM 5.1 *Dijkstra's Algorithm*

$p[s] := 0$,
$p[v] := B$ for all $v \neq s \in V$. (B is a big number, e.g. $B = \sum_{e \in E} w(e)$.)
$S := \emptyset$.
while $t \notin S$ **do**
    find a vertex $v \in V \setminus S$ such that $p[v] := \min\{p[u] \mid u \in V \setminus S\}$.
    $S := S \setminus \{v\}$.
    for $u \in V \setminus S$ such that $(v, u) \in E(G)$ **do**
        if $p[u] > p[v] + w(v, u)$ **then**
            $p[u] := p[v] + w(v, u)$,
            $p(u) := v$. (To keep the record of the shortest path.)
    end-for
end-while
return $p[t]$ and $p(u)$ for $u \in S$ **if** $t \in S$.

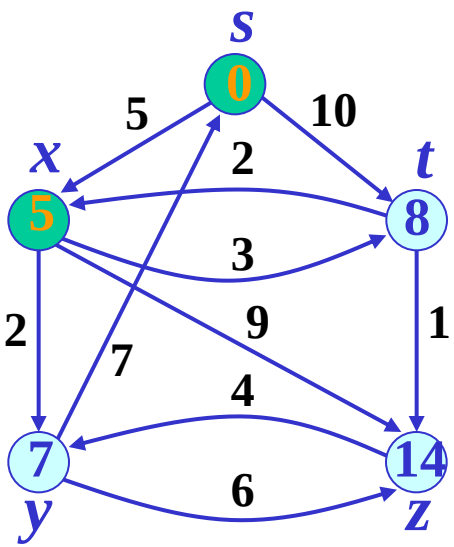
5.2 最短路（续五）



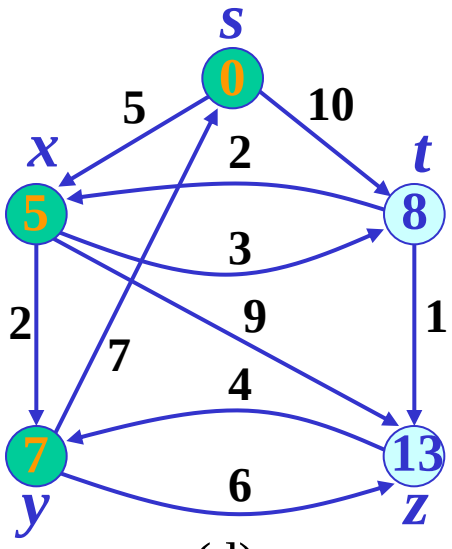
(a)



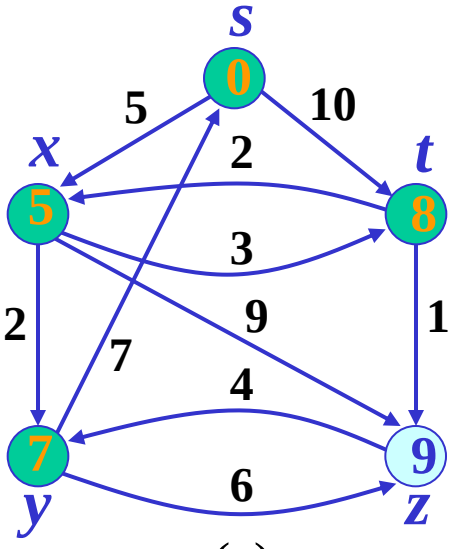
(b)



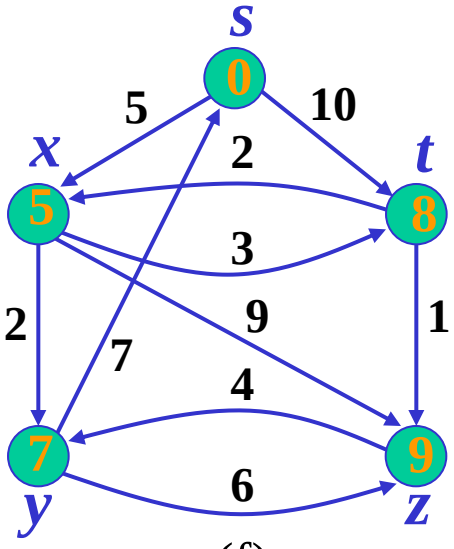
(c)



(d)



(e)



(f)

5.2 最短路（续六）

在**for-循环**语句内对边 $(v, u)\in E$ 进行的运算称为**松弛**。 这是因为它松弛了条件 $p[u] \le p[v] + w(v, u)$，该条件应被满足如果 $p[v]=p(s, v)$ 且 $p[u]=p(s, u)$。亦即，如果 $p[u]\le p[v]+w(v, u)$，那么没有必要满足该条件，故该条件被松弛了。

引理 3 对边 $(v, u)\in E$ 进行松弛，可得 $p[u] \le p[v] + w(v, u)$。

引理 4 对每一个顶点 $v\in V$，都有 $p[v] \le p(s, v)$，且在对图 G 的边进行松弛时，它自始至终都成立。而且，一旦 $p[v]$ 达到了其下界 $p(s, v)$，它便不会再改变了。

练习： 证明 **引理 3-4**。

练习：任给一个边赋权图G 及顶点 s 和 t。设 u 是从 s 到 t 的最短路 $p(s, t)$ 上的一个节点，且 $u\neq s, t$。问 $p(s, t)$ 的长度是否必为 s 到 u 的最短路的长度与 u 到 t 的最短路的长度之和？

5.2 最短路（续七）

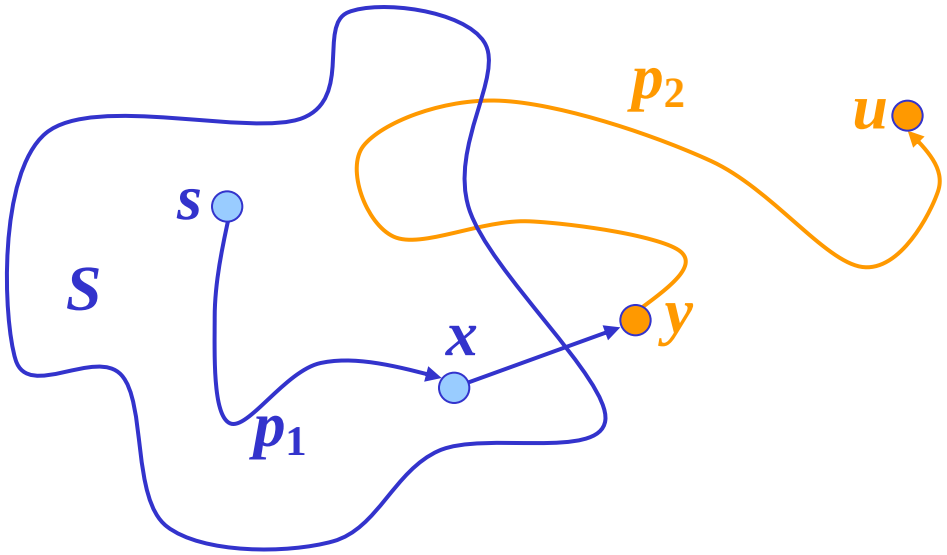
定理 1 **Dijkstra 算法**可以在时间$O(|V|^2)$内求解非负权值的两点之间**最短路问题**。

证明 要证明算法的正确性，我们仅需要证明，对每一个顶点$v \in V$，当 v 被放入集合 S 中时，有$p[v] = p(s, v)$，且在算法的运行过程中，此等式一直成立。

采用反证法。假设 u 是第一个被放入集合 S 中，且满足 $p[u] \neq p(s, u)$ 的顶点。现在我们分析一下，在 u 被放入 S 的 **for-**循环中的开始阶段的情形。此时有 $u \neq s$，这是因为 s 是第一个被放入 S 中的顶点，且 $p[s]=0$。由于 $u \neq s$，故可知，在 u 被放入 S 中之前，有 $S \neq \varnothing$。易知，一定存在由 s 到 u 的某条路，否则由**定理4** 可知 $p[s] = \infty$，这与 假设 $p[u] \neq p(s, u)$ 产生了矛盾。

5.2 最短路（续八）

因为从 s 到 u 之间至少存在一条路，因而存在一条最短路 p。路 p 与 S 中的一个点相连，即 s，也与 $V\setminus S$ 中一个点相连，即 u。 让我们考虑 p 上的第一个顶点 y 其满足 $y\in V\setminus S$，并令 $x\in V$ 是路 p 上 y 的前一个顶点。因而，路 p 可以分解为两条路，由 s 到 x 的路 p_1 和由 y 到 u 的路 p_2，这两条路由边 (x, y) 相连。



5.2 最短路（续九）

现在断言：当顶点 u 被放入集合S中时， $p[y] = p(s, y)$。 若不然，$x \in S$。且 u 是第一个这样的顶点：当它被放入 S 时，有 $p[u] \neq p(s, u)$。因此，当 x 被放入 S 时，有 $p[x] = p(s, x)$。 此时，边 (x, y) 被松弛了，由 **定理 3-4** 即得断言。

因为， y 在由 s 到 u 的最短路上，且它比 u 要早出现；此外，所有的边的权重都是非负的，因而有 $p(s, y) \leq p(s, u)$。 故由 **引理4** 可得

$$p[y] = p(s, y) \leq p(s, u) \leq p[u]。$$

然而，当选取 u 时，u 和 y 都属于 $V \setminus S$，故有 $p[u] \leq p[y]$。因有

$$p[y] = p(s, y) = p(s, u) = p[u]。$$

因而可得 $p[u] = p(s, u)$，与顶点 u 的选取相矛盾。由此可知，每一次当顶点 $u \in V$ 被放入 S 时，我们都有 $p[u] = p(s, u)$，再由 **引理 4**，可知这个等式以后一直成立。

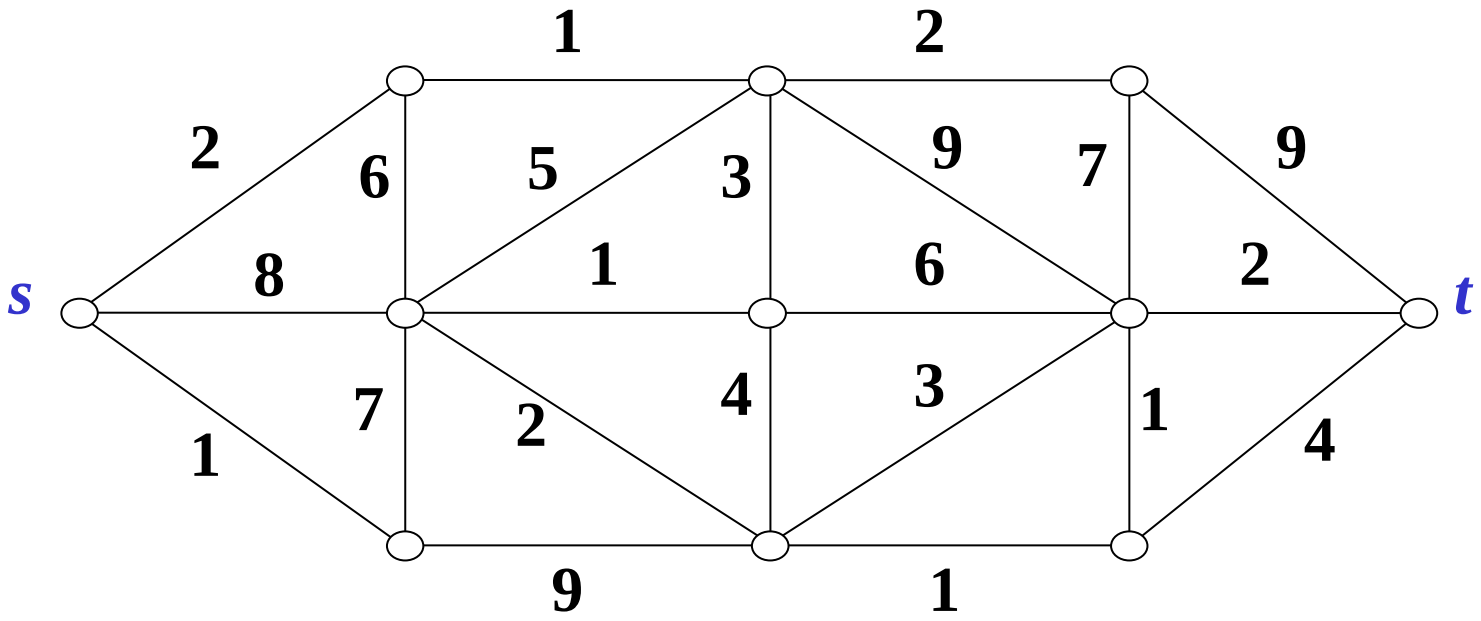
5.2 最短路（续十）

要论证一旦 $d[u] = p(s, u)$， $p[u]$ 再也不会改变了。注意，在 $p[u]$ 达到它的下界以后，它就不可能再变小了，这是因为我们刚刚证明 $p[u] \geq p(s, u)$，而且它也不会增加了，这是因为**for-循环**语句中的运算步骤不会增加 $p[\ldots]$ 的值。

最多有 $|V|$ 个顶点被放入集合 S 中，且它们被放入 S 中恰好一次，因此， **while-循环**中最多有 $|V|$ 次迭代。在每一次迭代中，在一个顶点被放入 S 中以前，最多有 $|V|$ 个顶点被考察过。因而，总共有 $O(|V|^2)$ 次运算。此外，在**for-循环**中 E 中每一条边最多被考虑了一次。由此可知，整个算法的运行时间是 $O(|V|^2 + |E|) = O(|V|^2)$。

5.2 最短路（续十一）

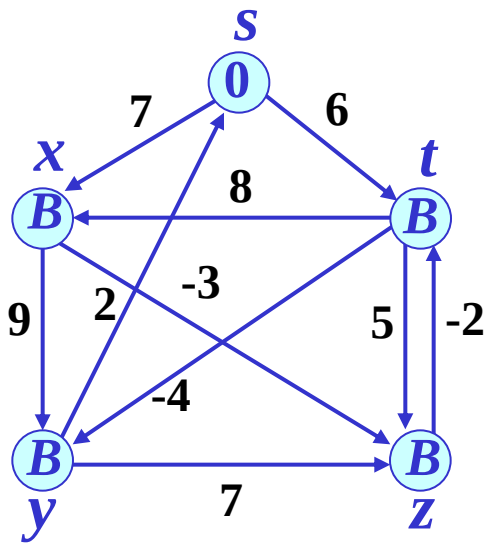
练习题：求出图 ***G*** 中顶点 ***s*** 到顶点 ***t*** 之间的最短路的长度。



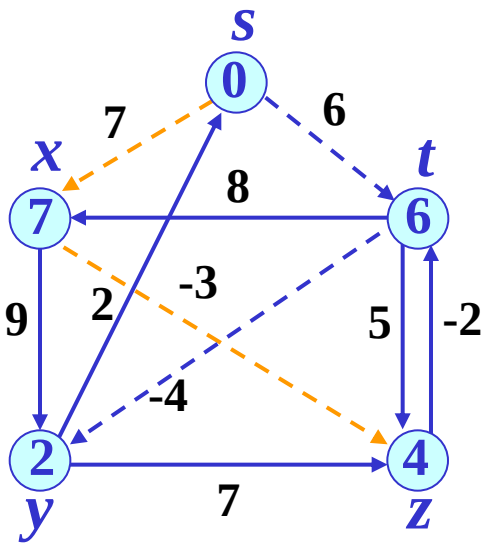
思考题：任给一个边赋权图***G***，及顶点 ***s*** 和顶点 ***t***。设 ***u*** 是从 ***s*** 到 ***t*** 的最短路 ***p*(*s*, *t*)** 上的一个节点，且***u*≠*s***，***u*≠*t***。问： ***p*(*s*, *t*)** 的长度是否一定等于 ***s*** 到 ***u*** 的最短路的长度与 ***u*** 到 ***t*** 的最短路的长度之和？为什么？

5.2 最短路（续十二）

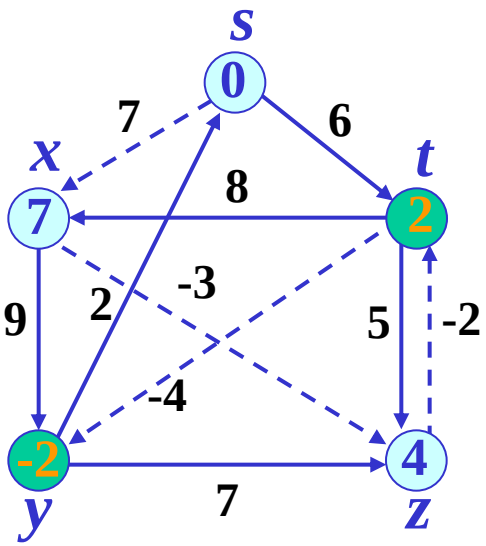
最后，我们考虑更一般的情形：所给图上的边的权重可以是负数。在此情形下，**Dijkstra 算法** 有可能找不到最短路。下面图示就给出了这样一个实例。



有负权重的有向图



Dijkstra算法找到的一条路



最短路：$p(s, t)=2$，$p(s, y)=-2$.

5.2 最短路（续十三）

对于一般情况（存在负权重），可以应用**Bellman–Ford–Moore 算法**（1958年，1956年，1957年）求两点之间的最短路。 确切地说，该算法可以判断给定的图中是否存在一个圈其值为负数，且从源点可以到达此圈。如果确实存在这样一个圈，那么算法说明两点之间不存在最短路；如果不存在这样一个圈，那么算法会输出两点之间的一条最短路及其长度。

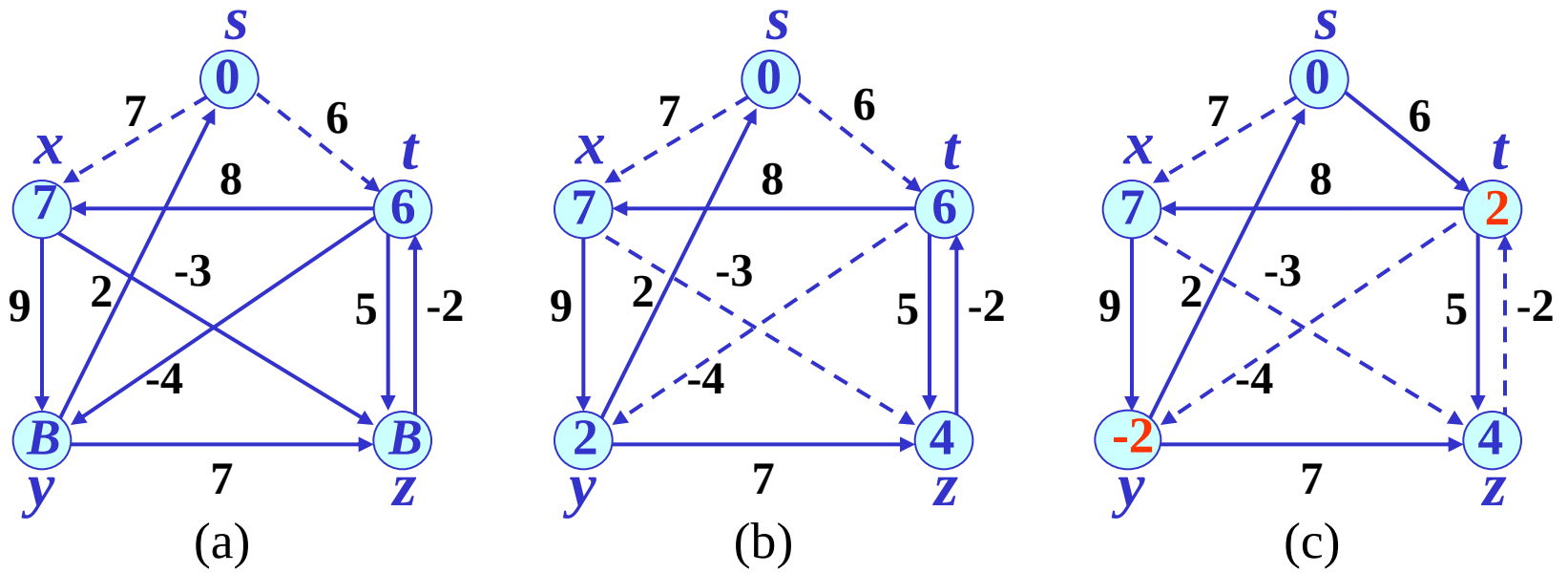
与**Dijkstra算法**类似， **Bellman–Ford–Moore 算法**也采用松弛运算，逐步地减少从 s 到每一个顶点 $v \in V$ 的最短路的长度 $p[v]$，直到它等于真正最短路的长度 $p(s, v)$。

5.2 最短路（续十四）

ALGORITHM 5.2 *Bellman-Ford Algorithm*

$p[s] := 0$,
$p[v] := B$ for all $v \neq s \in V$. (B is a big number, e.g. $B = \sum_{e \in E} w(e)$.)
$S := \emptyset$.
for $i := 1$ **to** $|V| - 1$ **do**
    for each $(v, u) \in E(G)$ **do**
        if $p[u] > p[v] + w(v, u)$ **then**
            $p[u] := p[v] + w(v, u)$,
            $p(u) := v$. (To keep the record of the shortest path.)
    end-for
end-for
for each $(v, u) \in E(G)$ **do**
    for each $(v, u) \in E(G)$ **do**
        if $p[u] > p[v] + w(v, u)$ **then**
            return false.
    end-for
return *true* along with $p[t]$ and $p(u)$.

5.2 最短路（续十五）



上面的图示给出了 **Bellman–Ford–Moore 算法**求解前面的实例的过程。可以看出， 在 **4** 个对边的相继运算中的每一个情况。此时，不存在负权重的圈，因而，算法给出了最短路，在图(c) 中用虚线表示。

5.2 最短路（续十六）

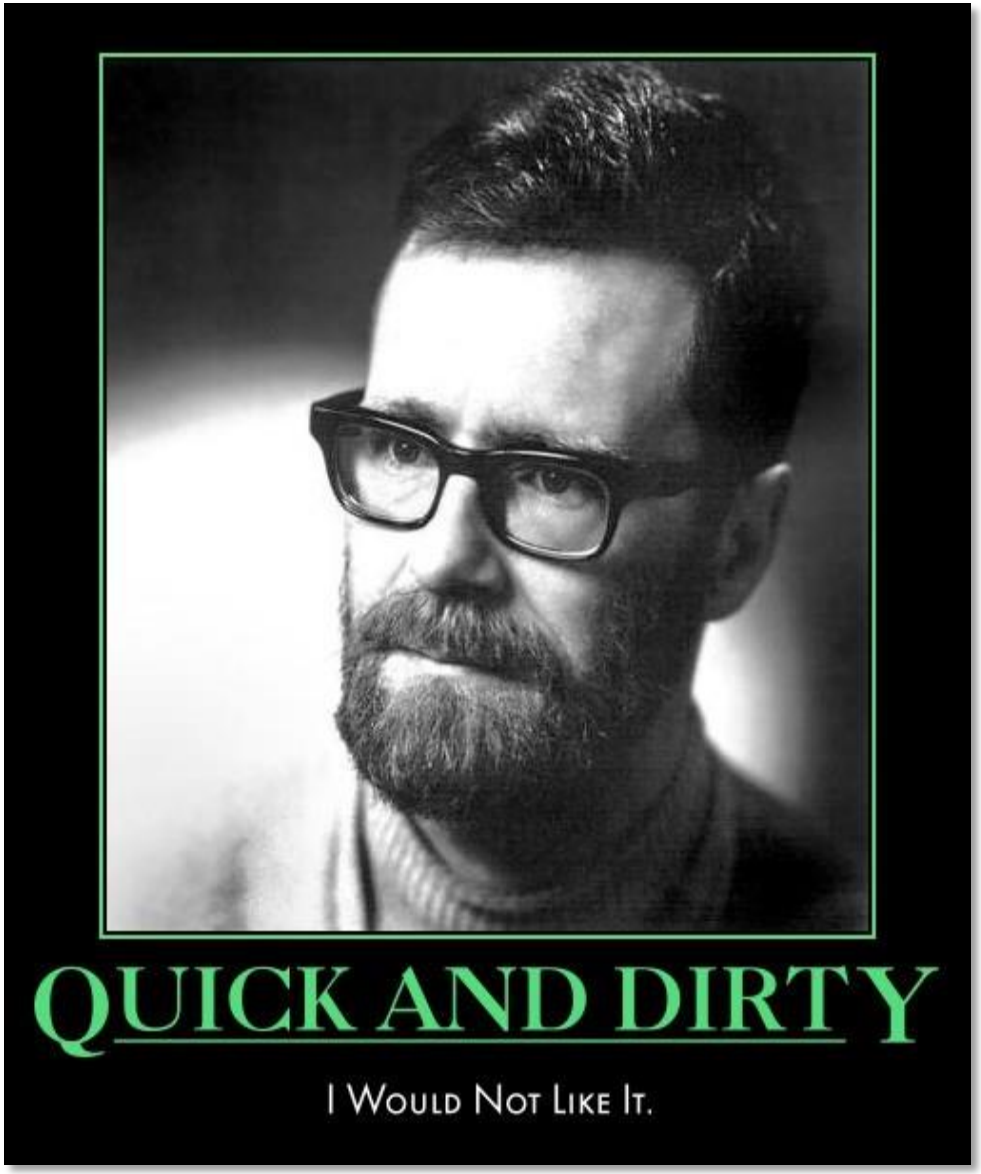
注意，在上述算法中，在进行完通常的初始化以后，它会对给定图的边进行 $|V|-1$ 次运算。每一次是在外**for-循环**里面的内**for-循环** 的一次迭代，它包括对图的每一条边松弛一次。此后，它用最后的 **for-循环**检查是否存在负圈，并报告存在负圈或者输出最短路。回忆一下，在**Dijkstra算法**中， 每一条边 (u, v) 恰好被松弛一次，以试图改进从源点经顶点 u 到顶点 v 的最短路；而在 **Bellman–Ford–Moore 算法**中，每一条边会被松弛多次。

最后，需要指出的是，当给定的图 G 存在负圈时，人们还未发现求最短路的多项式时间算法。实际上，此问题是 **NP-难解**的。

思考题：是否可以用动态规划原理和方法，设计一个求解最长路的多项式时间算法？为什么？

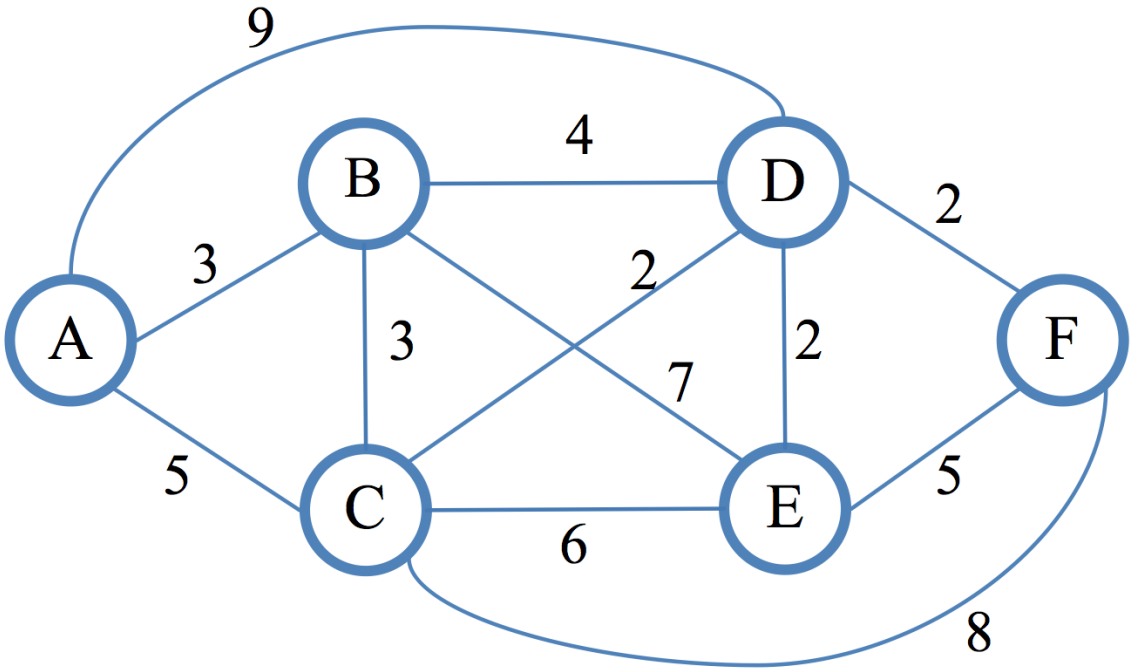
5.2 最短路（续十七）江宏，LeanCloud 联合创始人/CEO

Dijkstra是几位影响力最大的计算科学的奠基人之一，也是少数同时从工程和理论的角度塑造这个新学科的人。他的根本性贡献覆盖了很多领域，包括：编译器、操作系统、分布式系统、程序设计、编程语言、程序验证、软件工程、图论等等。他的很多论文为后人开拓了整个新的研究领域。我们现在熟悉的一些标准概念，比如互斥、死锁、信号量等，都是**Dijkstra** 发明和定义的。



5.2 最短路（续十七）江宏，LeanCloud 联合创始人/CEO

在**Edsger Wybe Dijkstra**决定成为一个程序员后，他尽快完成了学业，因为他在大学里不再受欢迎了：物理学家们觉得他是逃兵，而**数学家们也看不起他和他做的事，因为在当时的数学文化里，你的课题必须和 ∞ 有关才会受尊重**。那个时候程序设计没有成为一个职业。1957 年，他结婚时在申请栏写上了「程序员」，结果被政府拒绝，因为当时荷兰没这个职业。



5.2 最短路（续十八）江宏，LeanCloud 联合创始人/CEO

有一天**Dijkstra**和未婚妻在阿姆斯特丹购物，他们停下来在一家咖啡店的阳台上喝咖啡休息，他开始思考这个问题。他觉得可以让计算机演示如何计算荷兰两个城市间的最短路径，这样问题和答案都容易被人理解。于是他在 20 分钟内想出了高效计算最短路径的方法。**Dijkstra** 自己也没有想到这个 20 分钟的发明会成为他最著名的成就之一，并且会被以他的名字命名为 **Dijkstra** 算法。

三年以后这个算法才首次发布，但**当时的数学家们都不认为这能成为一个数学问题：两点之间的路径数量是有限的，其中必然有一条最短的，这算什么问题呢？**

Dijkstra 的眼科医生有一天突然问他：「是你发明了 GPS 导航的算法吗？」。一问之下，原来他读了 2000 年 11 月的科学美国人杂志，讲 GPS 的文章里说到了 **Dijkstra**。

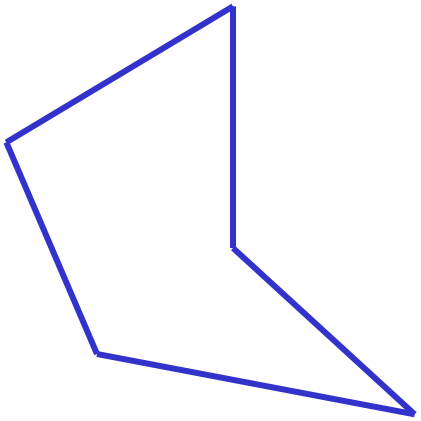
5.2 最短路（续十九）江宏，LeanCloud 联合创始人/CEO

1960 年代后期，由于计算机变得越来越强大，程序设计和维护的方式跟不上软件复杂度的快速上升，世界进入了「软件危机」。**Dijkstra** 在 ACM 的月刊上发表了一篇名为 **GOTO Statement Considered Harmful** 的文章为全世界的程序员们指明了方向，这就是结构化程序设计运动的开始。他和 Hoare、Dahl 合著的《结构化程序设计》成为了这次软件史上第一次变革的纲领，影响了此后大部分程序设计语言，包括 70、80 后程序员熟悉的 C 和 Pascal。

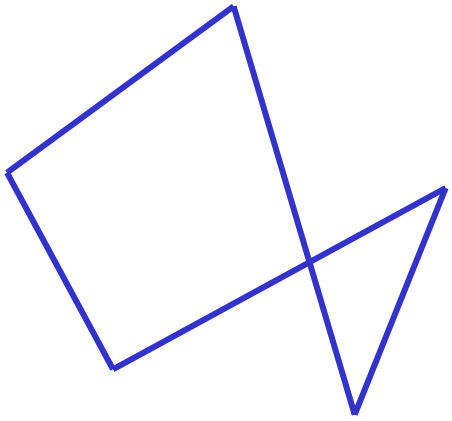
尽管计算机软件技术有很大一部分是 **Dijkstra** 发明的，但他却很少使用计算机，**或许这和他作为程序员时很大一部分时间是在为还没造出来的计算机开发程序有关系。**

5.2 最小多边形剖分

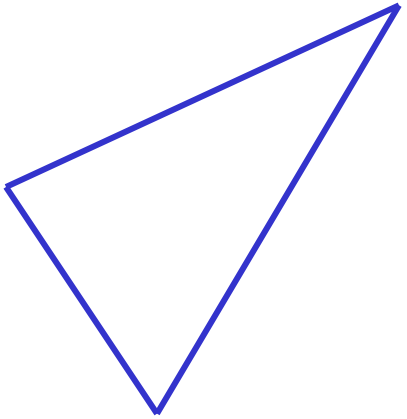
下面我们考虑如何对一个凸多边形进行最优的三角剖分。一个**多边形**是平面上一个分段线性的闭曲线。换言之，它是有一系列直线段，称为多边形的**边**，组成了封闭曲线。称一个多边形为**三角形**如果它只有三条边。 连接两条相邻边的点称为多边形的**顶点**。



一个多边形



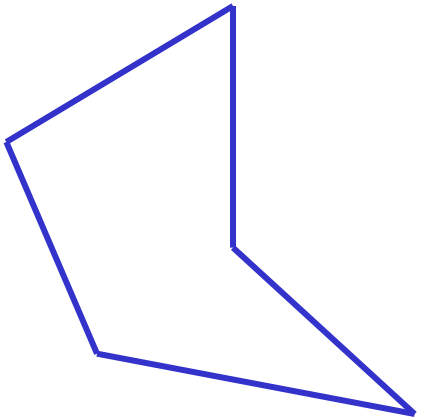
一个多边形



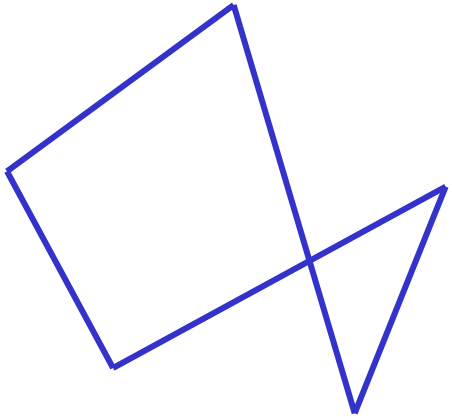
一个三角形

5.2 最小多边形剖分（续一）

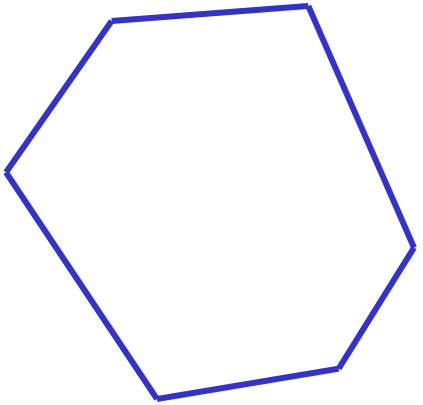
这里我们仅考虑简单多边形，即不存在交叉的两条边的多边形。平面上的一个简单多边形内部的点称为内点，多边形边上的点称为外点。 我们称一个简单多变形为凸的，如果任意两个内点或者外点的连线上的点也是简单多边形的内点或者外点。



非凸多边形



有交叉的两条边的多边形

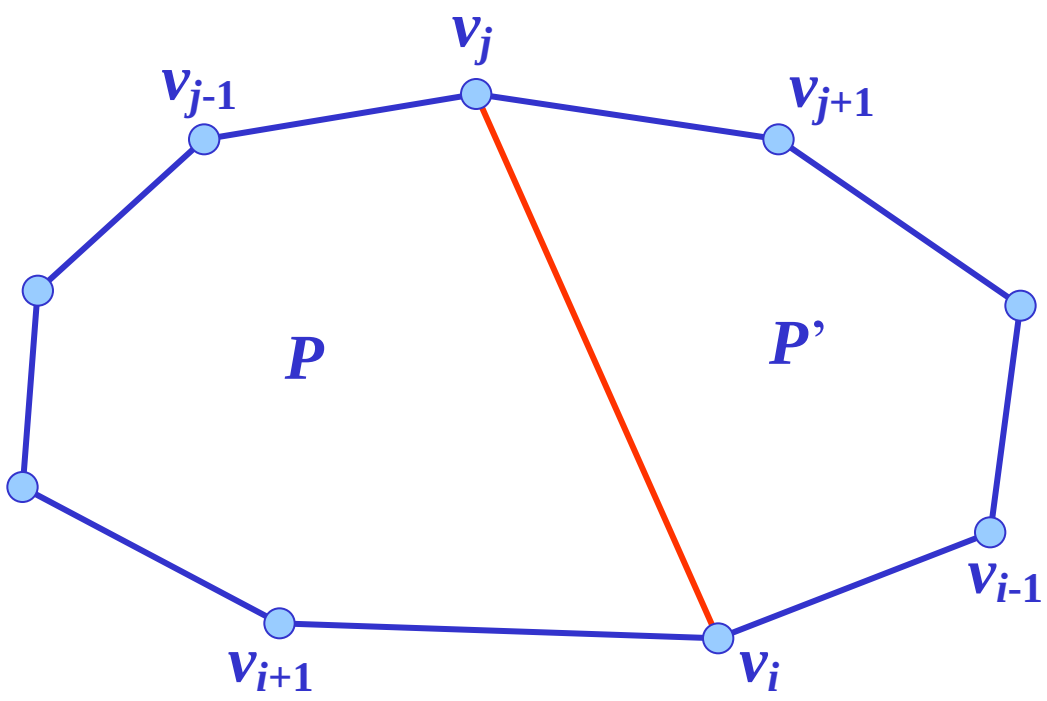


简单多边形

5.2 最小多边形剖分（续二）

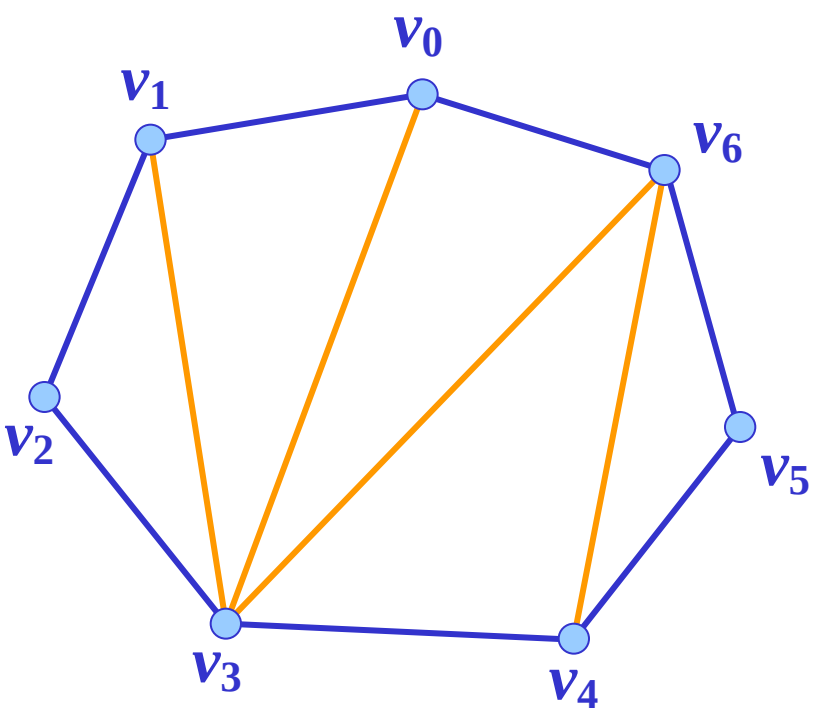
任给两个非相邻的顶点 v_i 和 v_j， 线段 $v_i v_j$ 是多边形的一条弦。弦 $v_i v_j$ 将多边形划分成两个子多边形：

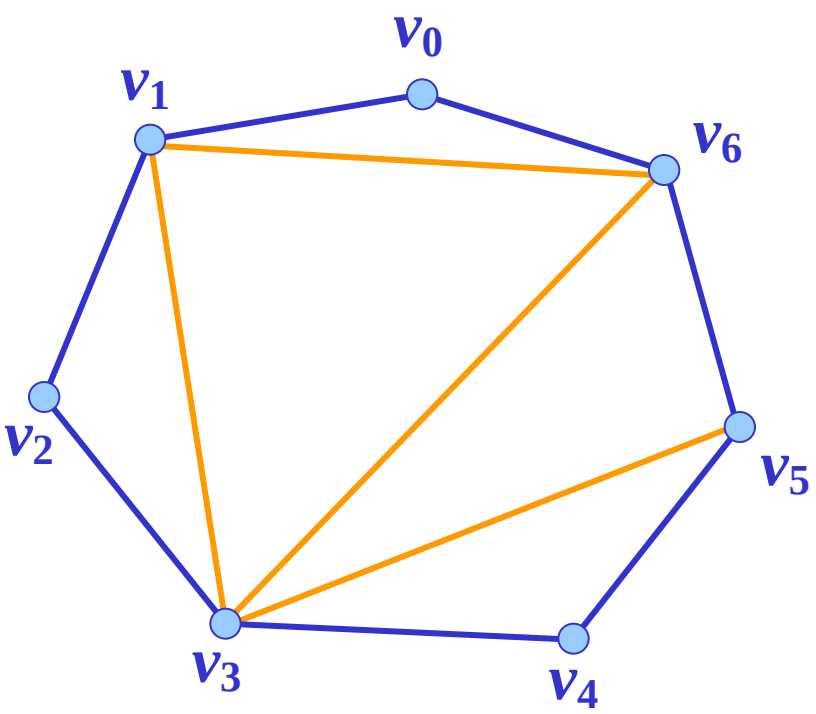
$P = < v_i, v_{i+1}, \ldots, v_j >$ 和 $P' = < v_j, v_{j+1}, \ldots, v_i >$.



5.2 最小多边形剖分（续三）

一个多边形的三角剖分 T 是由它的一组弦构成，它们将多边形划分为若干个互补相交的三角形。





5.2 最小多边形剖分（续四）

在一个三角剖分 T 中， 任意两条弦都不在端点以外的点相交，而且 T 中的弦是极大的：不在 T 中的任意一条弦都与 T 中的某条弦相交。 每一个具有 n-顶点的凸多边形的三角剖分都有 $n-3$ 条弦，它们将多边形分成 $n-2$ 个三角形。

问　题： 多边形的最小三角剖分

实　例： 凸多边形 $P=(v_0, v_1, \ldots, v_{n-1})$，一个定义在多边形的边和 P 的弦所形成的三角形的权重函数 w 。

可行解： 多边形 P 的一个三角剖分。

目　标： 最小化三角剖分所形成的三角形的权重之和。

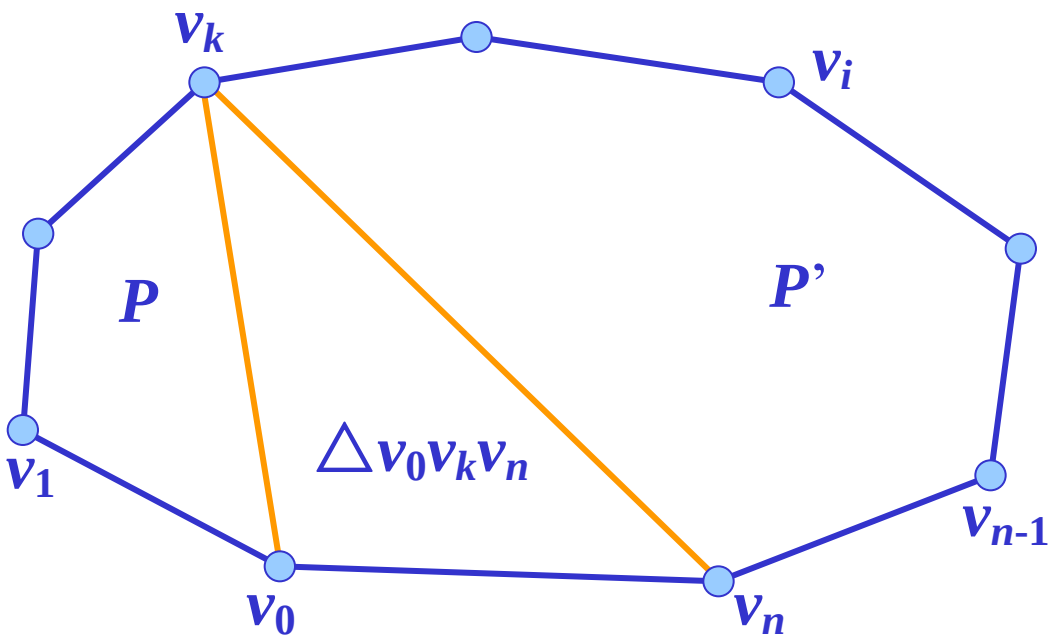
如下给出了三角形的权重函数的一个非常自然的定义：

$$w(\triangle v_i v_j v_k) = |v_i v_j| + |v_j v_k| + |v_k v_i|,$$

其中 $|v_i v_j|$ 是两点 v_i 和 v_j 之间的欧氏距离。

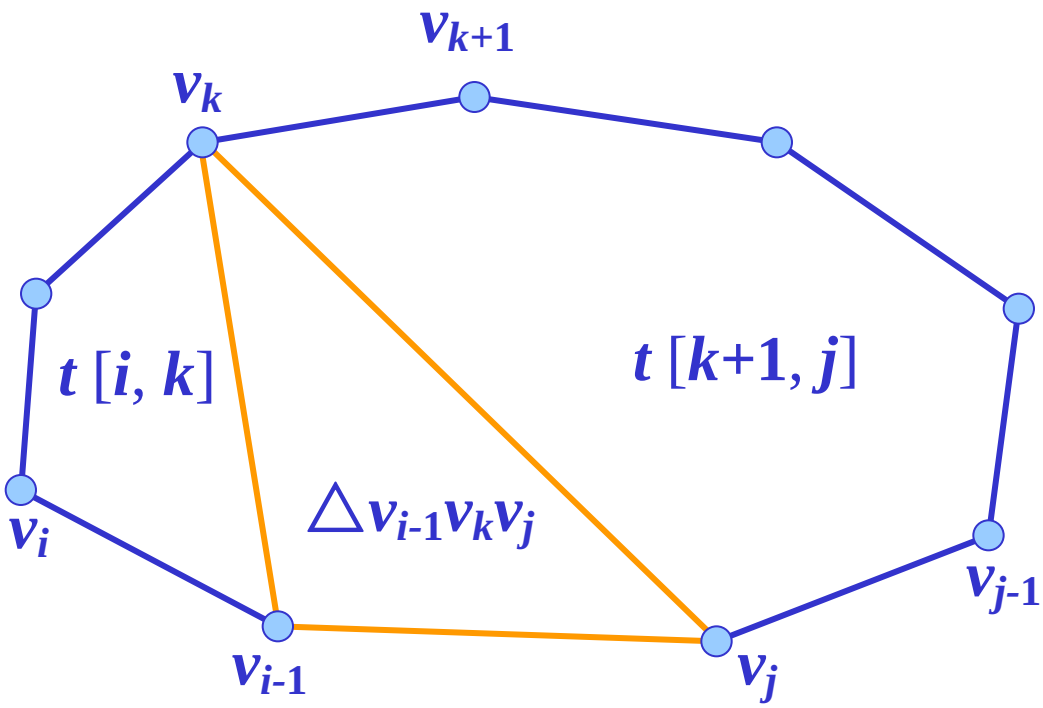
5.2 最小多边形剖分（续五）

要研究最优三角剖分的结构，假设T_{opt}是一个$(n+1)$-顶点的多边形 $P = < v_0, v_1, \ldots, v_n >$ 的一个三角剖分。不妨设它包含三角形 $\triangle v_0 v_k v_n$，其中 $1 \le k \le n-1$。显然，T_{opt} 的权重等于三角形$\triangle v_0 v_k v_n$ 的权重，与子多边形 $<v_0, v_1, \ldots, v_k>$ 和 $< v_k, v_{k+1}, \ldots, v_n >$ 的权重之和。这样由 T_{opt} 产生的两个子多边形的三角剖分也分别是最优的。

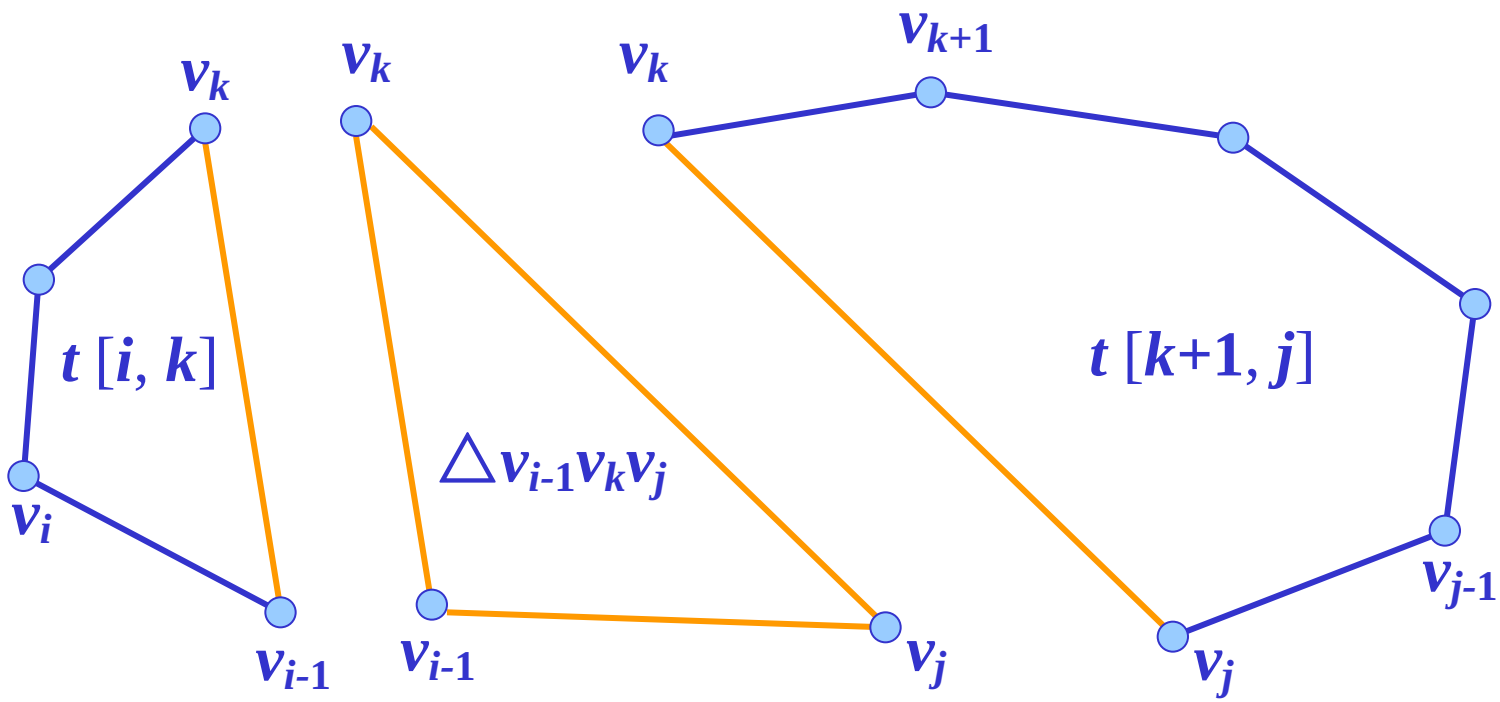


5.2 最小多边形剖分（续六）

下面我们用 $t[i, j]$， $1 \le i \le j \le n$，表示多边形 $<v_{i-1}, v_i, \ldots, v_j>$ 的最优三角剖分。



5.2 最小多边形剖分（续七）



设计动态规划算法的下一步，就是给出 $t\,[i, j]$ 的递归式：

$$t\,[i, i] = 0，\ i=1, 2, \ldots, n.$$

5.2 最小多边形剖分（续八）

当 $j-i \geq 1$ 时， 多边形 $<v_{i-1}, v_i, \ldots, v_j>$ 至少含有 3 个顶点。我们想优化选取 v_k，$k = i, i+1, \ldots, j-1$，使得三角形$\triangle v_{i-1}v_kv_j$的权重、最优三角剖分$<v_{i-1}, v_i, \ldots, v_k>$ 和 $<v_k, v_{k+1}, \ldots, v_j>$ 的权重之和最小。 这可以用如下的递归关系式表述：

$$t[i, j] = \min\{\, t[i, k] + t[k+1, j] + w(\triangle v_{i-1}v_kv_j) \mid i \leq k \leq j-1 \},\ i < j。$$

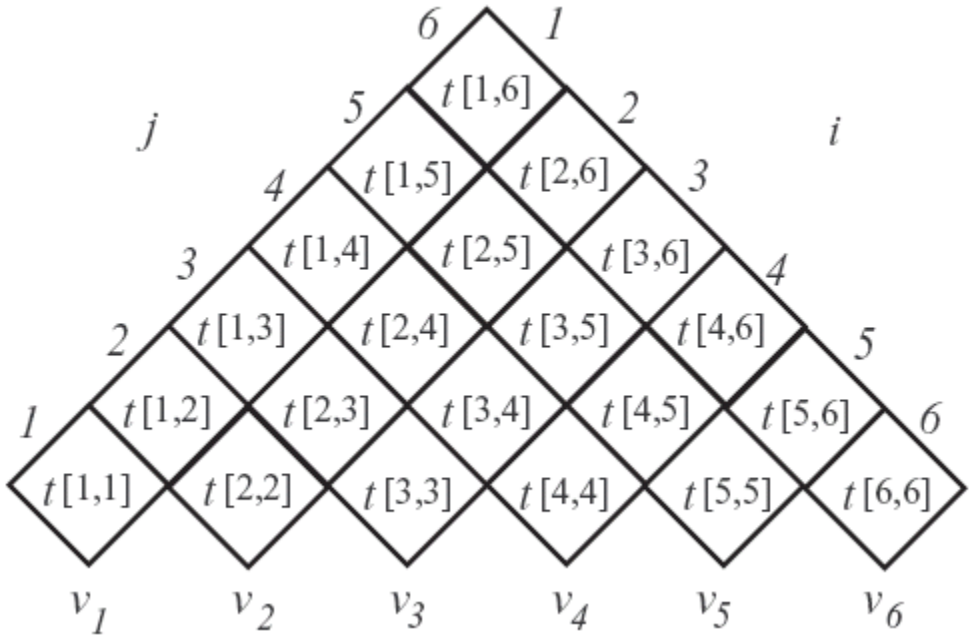
为了记得，我们是如何一步一步地构造一个最优三角剖分，我们用 $s[i, j]$ 表示指标 k，它使得可以划分多边形 $< v_{i-1}, v_i, \ldots, v_j >$，并得到一个最优三角剖分。

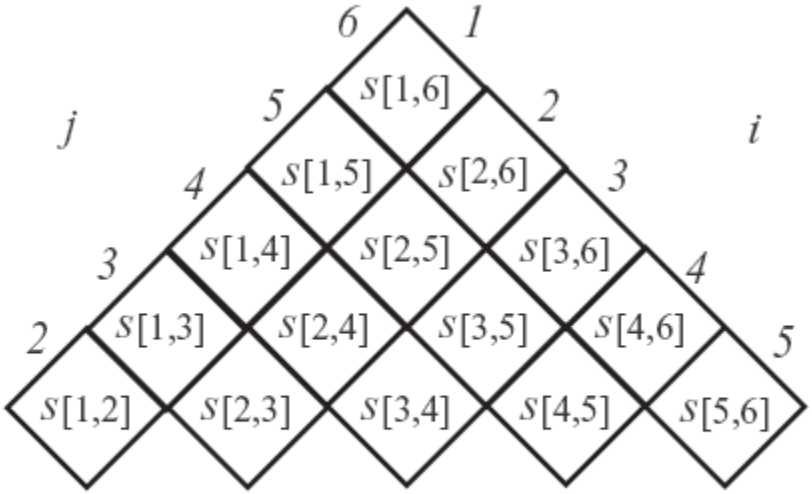
5.2 最小多边形剖分（续九）

ALGORITHM 5.3 *Polygon Triangulating Algorithm*

$t[i,i] := 0$ for $i = 1, 2, \cdots, n$.
for $l := 2$ **to** n **do**
  for $i := 1$ **to** $n - l + 1$ **do**
    $j := i + l - 1$,
    $t[i,j] := \infty$.
    for $k := i$ **to** $j - 1$ **do**
      $q := t[i,k], +t[k+1,j] + w(\triangle v_i v_k v_j)$.
      if $q < t[i,j]$ **then**
        $t[i,j] := q$,
        $s[i,j] := k$.
    end-for
  end-for
end-for
return $m[i,j]$ and $s[i,j]$.

5.2 最小多边形剖分（续十）

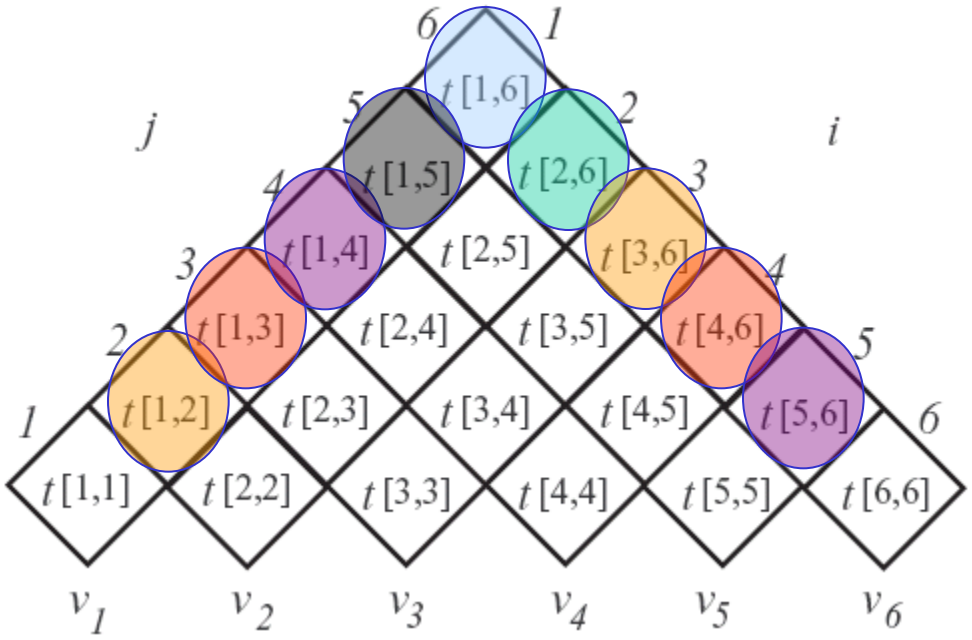


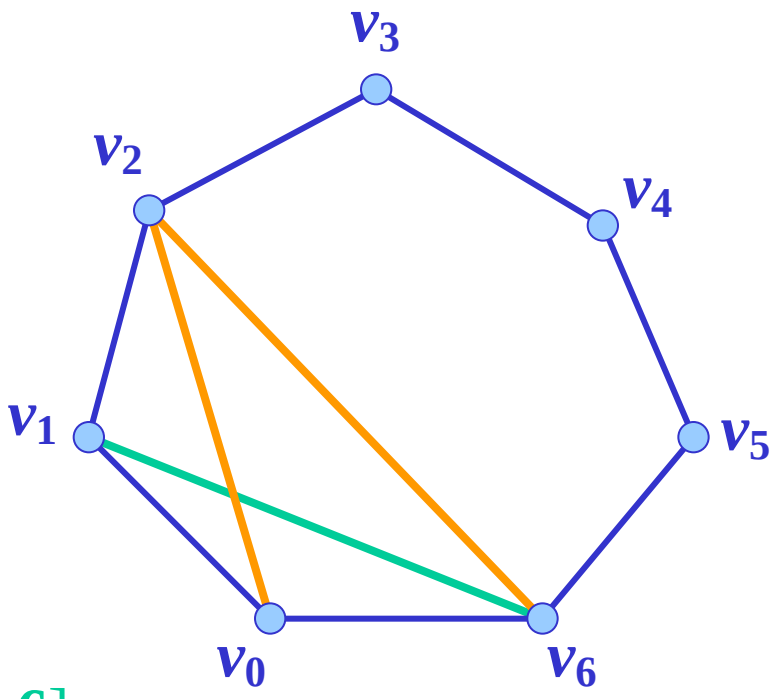


$t[i, i] = 0$，$i=1, 2, \ldots, n$。

$$t[i, j] = \min\{\, t[i, k] + t[k+1, j] + w(\triangle v_{i-1}v_k v_j) \mid i \le k \le j-1 \}, \; i < j。$$

5.2 最小多边形剖分（续十一）





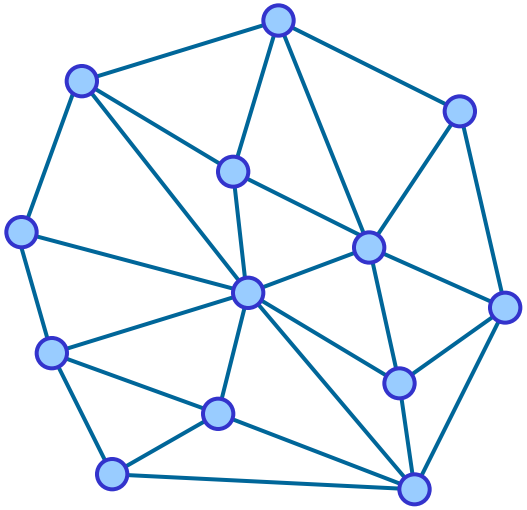
$$t[1,6]=\min\{\ w(\triangle v_0v_1v_6)+t[2,6],$$
$$t[1,2]+w(\triangle v_0v_2v_6)+t[3,6],$$
$$t[1,3]+w(\triangle v_0v_3v_6)+t[4,6],$$
$$t[1,4]+w(\triangle v_0v_4v_6)+t[5,6],$$
$$t[1,5]+w(\triangle v_0v_5v_6)\ \}$$

5.2 最小多边形剖分（续十二）

定理 2 动态规划方法可以在 $O(n^3)$ 时间内找到凸多边形的一个最优三角剖分。

证明. 练习。

另一个三角剖分问题是 - **最短三角剖分问题**：任给欧氏空间上的 n 个点，如何构造这 n 个点的一个三角剖分使得每条边的离散欧氏长度之和最短。该问题**NP-难**的。当给定的 n 个点可以构成一个 n 个点的多边形时，此问题就是上面讨论的问题了。



练习. 证明：欧氏空间上 n 个点的任意一一个三角剖分都包含 $2n - 3 + k$ 条边，其中 k是这 n 个点所形成的凸包中给定点的个数。(左图：n =13，k = 5，三角剖分含16个三角形，共$2n - 3 + k$=28条边。)

5.2 背包

最后，我们讨论著名的背包问题。 假设你计划进行一次旅行，为此你需要在 n 个物品中挑选一些基本物品随身带着。每一个物品都有一个体积，占用一定的空间，你随身带的背包有一个容量限制。你可以将每一件物品都赋予一个数值，代表你带上它可以为你产生的好处。你的目标是：如何选取若干个物品并把它们放入背包中，使得你得到的好处最多。这里假定，你不能带分数件物品，亦即你或者带或者不带。

问　题：背包

实　例：n 件物品 $X=\{x_1, \ldots, x_n\}$，物品 x_i 可产生效益 $p_i \in Z^+$，但占用空间 $s_i \in Z^+$，容积上限 $c \in Z^+$。

可行解：若干件物品 $Y \subseteq X$ 使得 $\sum_{x_i \in Y} s_i \leq c$。

目　标：最大化 Y 中物品的效益之和 $\sum_{x_i \in Y} p_i$。

5.2 背包（续一）

为了应用动态规划方法设计求解背包问题的一个算法，我们需要引入子问题的概念，使得可以明确地刻画最优性。为此，我们考虑如下问题：求物品的一个子集合 $\{x_1, \ldots, x_k\}$ 使得在效益之和为 p 且占用总空间不超过 c 的集合中，它占用的总空间最小

$0 \le p \le P \equiv \sum_{i=1}^{n} p_i$，其中 $1 \le k \le n$。

我们用 $c(k, p)$ 表示这个新问题的最优解，$s(k, p)$ 表示最优解占用空间的大小。另外，我们假定，若 $c(k, p)$ 无定义，则令

$s(k, p) = 1 + \sum_{i=1}^{n} s_i$。

根据上述定义，显然有，$c(1,0) = \varnothing$，$c(1, p_1) = \{x_1\}$，而 $c(1, p)$ 没有定义当 $p \neq p_1$。此外，令 p^* 为最大的指标 p 使得 $c(n, p) \neq \varnothing$，则 $c(n, p^*)$ 是原问题的最优解。

5.2 背包（续二）

注意，总效益为 p 的最优子集合 $\{x_1, \ldots, x_k\}$ 或者是总效益为 $p-p_k$ 的 $\{x_1, \ldots, x_{k-1}\}$ 的最优子集合加上物品 x_k，或者是总效益为 p 的 $\{x_1, \ldots, x_{k-1}\}$ 的最优子集合。因而，我们可得如下的关系式：对于任意 k，$2 \le k \le n$，和任意 p，$0 \le p \le P$

$$c(k, p) = c(k-1, p-p_k) \cup \{x_k\},$$

如果下面的条件 (*) 成立，

(*) $\begin{cases} p_k \le p, c(k-1, p-p_k) \text{ 有定义，且} \\ s(k-1, p-p_k) + s_k \le s(k-1, p)，\text{且 } s(k-1, p-p_k) + s_k \le c。 \end{cases}$

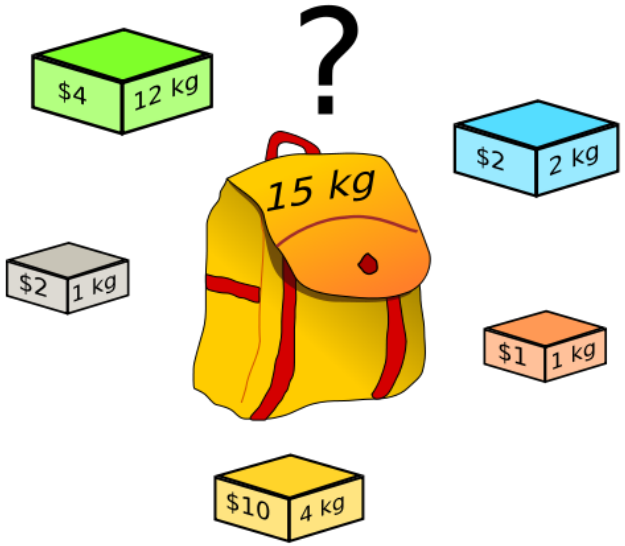
否则 (上述条件不成立)，

$$c(k, p) = c(k-1, p)。$$

5.2 背包（续三）

ALGORITHM 5.4 *Dynamic Programming Algorithm*

$c(1,p) :=$ undefined for $0 \leq p \leq P$;
$s(1,p) := 1 + \sum_{i=1}^{n} s_i$ for $0 \leq p \leq P$.
$c(1,0) := \emptyset$ and $s(1,p) := 0$;
$c(1,p_1) := \{x_1\}$ and $s(1,p) := s_1$.
for $k = 2$ **to** n **do**
  for $p = 0$ **to** P **do**
    if condition (*) is satisfied **then**
      $c(k,p) := c(k-1, p-p_k) \cup \{x_k\}$;
      $s(k,p) := s(k-1, p-p_k) + s_k$.
    else
      $c(k,p) := c(k-1,p)$;
      $c(k,p) := c(k-1,p)$.
  end-for - p.
end-for - k.
$p^* := \max\{p \mid c(n,p) \neq$ undefined $\}$;
return $c(n,p^*)$.



5.2 背包（续四）

定理 3 动态规划算法可以在 $O(nP)$ 步内找到背包问题的最优解，其所用时间不超过 $O(n^3 p_{\max} \log(c\, p_{\max}))$ 。

证明. 由最优性原理可知，动态规划算法可以找到最优解。为了估计算法的运行时间，注意，执行第一个和第二个**for-循环** 的主体都仅需要常数步。而第一个和第二个 **for-循环** 指标最多取到 P 而第三个 **for-循环** 指标最多取到 n。注意 P 不超过 $n\, p_{\max}$ 而问题的输入规模不超过 $O(n \log p_{\max}) + O(n \log c)$。

不过需要强调的是，运行时间是 nP 的多项式，而不是问题实例输入长度的多项式，亦即 $\sum \lceil \log p_i \rceil + \sum \lceil \log s_i \rceil + \lceil \log c \rceil$。

这种情况我们称算法的运行时间（或者简单地说，算法）是**伪多项式时间**的。