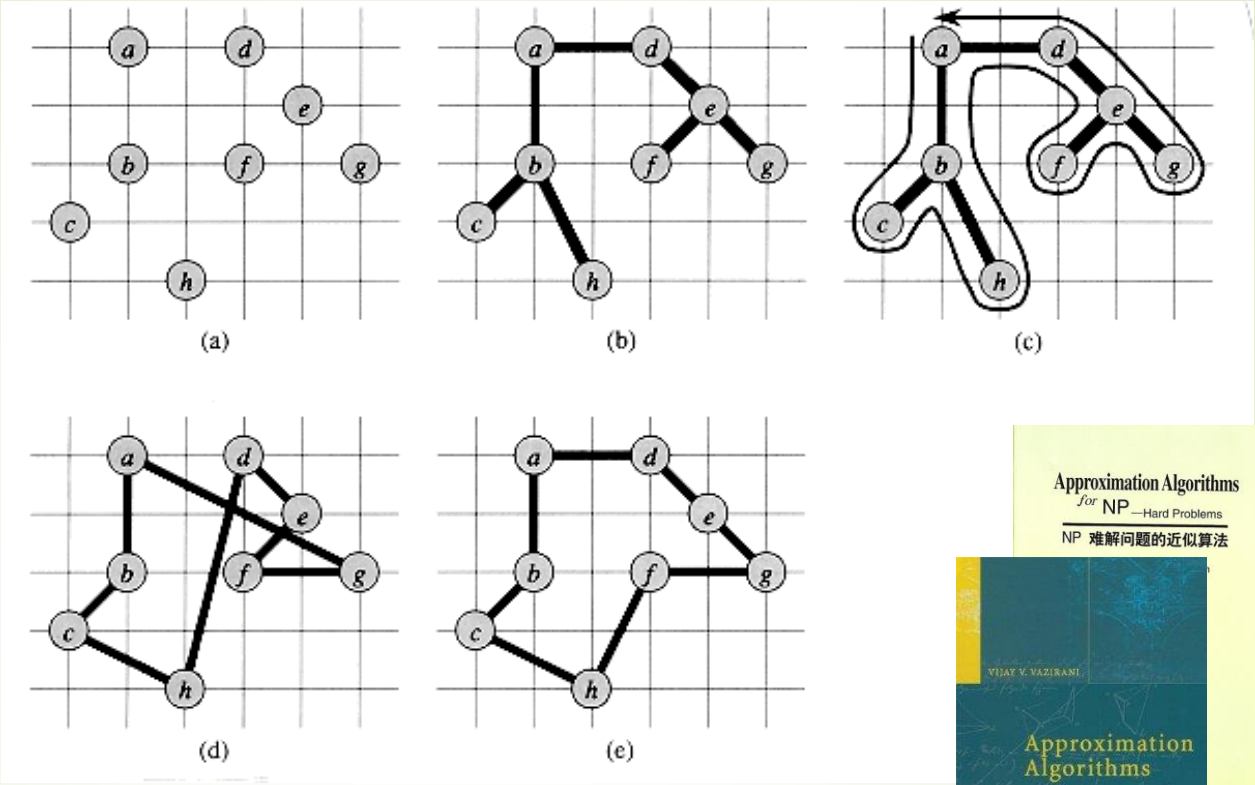
复习与回顾

最短路径

prim.swf



Approximation Algorithm
近似算法

Some Concepts

- NP – *Completeness*
  - (Nondeterministic Polynomial time)
- 近似算法

Exact algorithm vs. Heuristics

- Exact algorithm (精确算法)

  An algorithm that guarantees finding the (optimal) solution.

- Heuristics（启发式算法）

  A heuristic is “a technique which seeks **good** (i.e. near optimal) solutions at a **reasonable computational cost** without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is”.

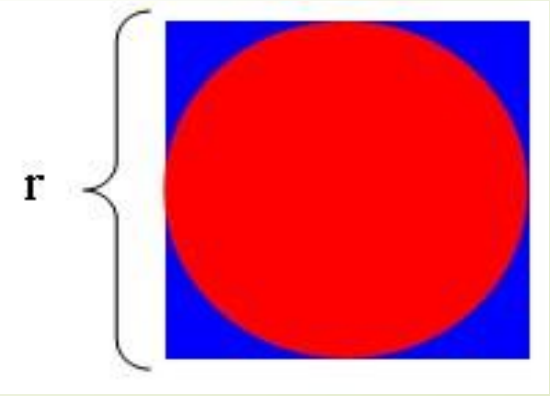
Deterministic vs. Probabilistic

- Deterministic algorithms（确定性）
  - If we run the algorithm twice with the same input, we get two identical execution patterns and results.
- Probabilistic algorithms（概率理论）
  - Depends also on some **random** events. The same algorithm may behave differently when it is applied twice to the same instance.

Probabilistic algorithms（概率理论）

- 举例：
  - 随机投点法计算pi值：

设有一半径为r的圆及其外切四边形。向该正方形随机地投掷n个点。设落入圆内的点数为k。由于所投入的点在正方形上均匀分布，因而所投入的点落入圆内的概率为(PI * pow(r,2)) / (4 * pow(r,2)) = PI / 4 。所以当n足够大时，k与n之比就逼近这一概率。从而，PI 约等于 (4*k)/n。



NP问题

例：要把 n 个数字以从大到小依次排列，则其计算量会因做法的不同而有相当的差别，一个直截了当的方法是，先求出最大的（比 (n-1) 次），再从不是最大的中间求次大的（比 (n-2) 次），再求第三大的（比 (n-3) 次），……如此一共比了

$$(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}$$

次就可以完成此工作。因此我们以 $O(n^2)$，即在 n^2 之层次来表此方法的计算量。

另外一种快排法，先把 n 个数分成若干小块，每块排好之后再合起来，则可以证明此种方法的计算量为 $O(n\log_2 n)$

如有一百万个数，则 $n^2=10^{12}$，而 $n\log_2 n$ 只有 2 x 10^7，其差别三个月与一分钟之比。

一般计算量的层次多以下表来区分：

$$O(\log n)<O(n)<O(n\log_2 n)<O(n^2)$$
$$<O(n^k)<O(2^n)<O(k^n)<O(n!)$$

NP问题

以计算机每秒作一百万次（10^6）计算为例

n	$\log n$	n	$n\log n$	n^2	n^3	2^n	3^n	$n!$
10	10^{-6}	10^{-5}	10^{-5}	10^{-4}	10^{-3}	10^{-3}	0.059	0.45
20	10^{-6}	10^{-5}	10^{-5}	10^{-4}	10^{-2}	1（秒）	58（分）	1年
50	10^{-5}	10^{-4}	10^{-4}	0.0025	0.125	36年	2×10^{10}年	10^{57}年
1000	10^{-5}	10^{-3}	10^{-3}	1	16小時	10^{333}年	極大	極大
10^6	10^{-5}	1	6	1月	10^5年	極大	極大	極大
10^9	10^{-5}	16小時	6天	3年	3×10^9年	極大	極大	極大

定义：凡对一个问题中最重要的参数n 而言，若能找到一个方法可以以方次上升的计算量完成， 我们称此问题为一 *P*-问题（*P* 为英文多项式 Polynomial 的第一字母）

NP问题举例

- 售货员旅行问题（traveling salesman problem）

  有一个售货员要开汽車到 n 个指定的城市去推销货物，他必须经过全部的 n 个城。現在他有一个 n 城的地图及各城之间的公路距离，试问他应如何取最短的行程从家中出发再到家中？

  计算复杂度：O(n!)

- 背袋问题

  有物体 n 个，各重 $w_1,w_2,\ldots,w_n$，今欲將它们分为2袋， 试问如何分法可使2袋之重量最为接近。

- 包装问题

  有 n 个各别重量小于 1 公斤的物品及足够可以装 1 公斤東西的盒子，今将物品装于盒子之中，多个物品可装于一盒，但任何一盒不得重于 1 公斤，试求最小的盒子数。

NP – *Completeness*

- Nondeterministic Polynomial time 非确定性的多项式时间
- *P* 是一个凡能用 $O(n^k)$计算量解决之问题的集合
- NP是P以外的问题集合(1971年古克,把 *P* 之外的问题归成了三大类，即 NP, NP-complete 及 NP-hard，古克定律)
- NP-Completeness问题：
  - 若有一个 NP-complete 问题可以用 $O(n^k)$ 计算量来解决，则全体的 NP 问题都可以用 $O(n^k)$ 之计算量来解决，即：若有一个 $x \in$ NP-complete且 $x \in$ P，则 *P*=NP。
  - 又换句话说，NP-complete 是 NP 中的难题，NP-complete 解决了， NP就解决了。但若有一个属于 NP 而不属于NP-complete 的问题解决了，则其他的 NP 问题不一定可以解决。

NP完全问题的近似算法

迄今为止，所有的NP完全问题都还没有确定的多项式时间算法。

对于这类问题，通常可采取以下几种解题策略。

(1) 只对问题的特殊实例求解

(2) 用动态规划法或分支限界法求解

(3) 用概率算法求解

(4) 只求近似解

(5) 用启发式方法求解

NP完全问题的近似算法

- 放弃寻求在多项式时间解决NP完全问题
- 近似解：不要求解决优化问题P的算法一定产生最优解，但一定要产生一个与最优解非常相近的可行解。
- 得到问题P的近似解的算法称为问题P的近似算法。

Approximation Algorithm（近似算法）

- 如果使用的算法所给出的输出仅仅是实际最优解的一个逼近，就会想知道这个近似最优解有多精确。
- 近似算法设计思想
  - 放弃求解最优解，用近似最优解代替最优解，以此换取：
    - 算法设计上的简化
    - 时间复杂性的降低
  - 近似算法是可行的：
    - 问题的输入数据是近似的
    - 问题的解允许有一定程度的误差
    - 近似算法可在很短的时间内得到问题的近似解

Approximation Algorithm（近似算法）

- 解非线性方程 $f(x)=0$:
  - 如果多项式的次数大于2，根的公式是怎样的呢？

    对于三次和四次多项式来说，它们的根公式是存在的，但却由于过于复杂，缺少实用价值。

  - 如果多项式的次数大于4，就没有只包含多项式系数、算术操作以及开根号操作的通用求解公式了。

Bisection method（二分法）

- 无法期望平分算法能够恰好发现方程的根并中止，需要一种标准来中止算法。
- 当括住某个根x^*的区间$[a_n,b_n]$变得足够小，以至于用x_n（区间的中点）逼近x^*的绝对误差肯定会小于某些预先选定的小数$\varepsilon>0$时，我们就可以把算法停止了。
- 因为x_n是$[a_n,b_n]$的中点，而x^*也位于这个区间，我们有

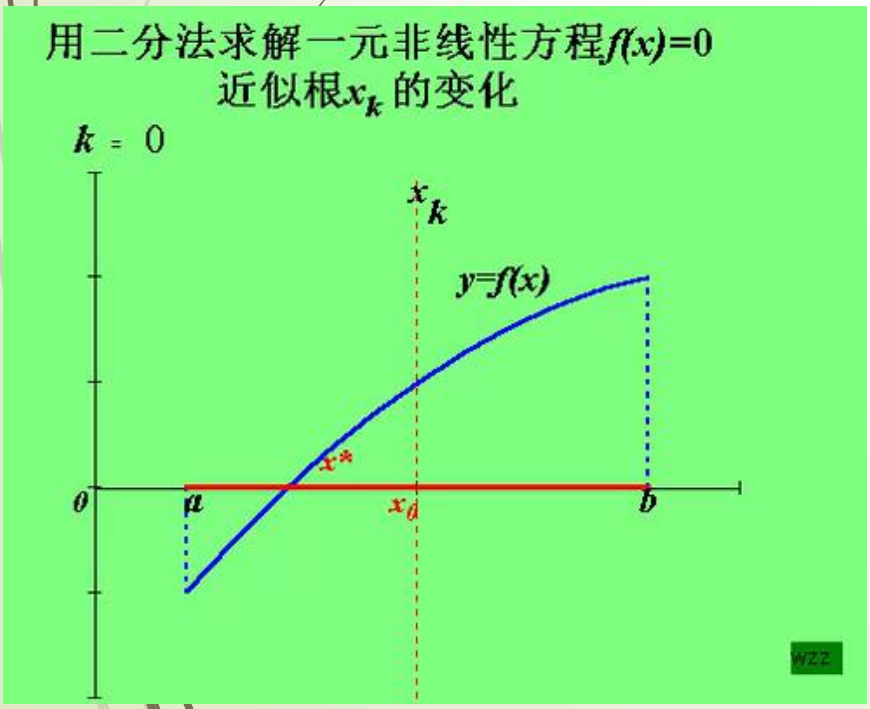
$$|x_n - x^*| \le \frac{b_n - a_n}{2}$$

- 因此，一旦（$b_n - a_n$）/2 $<\varepsilon$时，我们就可以停止该算法。
- 不难证明：

$$|x_n - x^*| \le \frac{b_1 - a_1}{2^n}$$

Bisection method（二分法）

- 可以用不等式事先求出能够满足要求的迭代次数，以获得预先设定的精确度；
- 可以选择一个足够大的迭代次数n，来满足不等式$(b_1-a_1)/2^n<\varepsilon$, 也就是，只要 $n>\log_2\frac{b_1-a_1}{\varepsilon}$ 即可



例：$x^3-x-1=0$
它有一个实根。因为$f(0)<0$而且f（2）>0，这个根一定位于区间（0，2）之间。
如果我们允许的误差级别是$\varepsilon=10^{-2}$， 需要$n>\log_2(2/10^{-2})$次，或者$n\geq 8$次迭代。

近似算法的性能

- 衡量近似算法性能的标准：
  - 时间复杂性：必须是多项式阶的——基本目标
  - 解的近似程度：——重要目标
- 若一个最优化问题的最优值为c*，求解该问题的一个近似算法求得的近似最优值为c，则将该近似算法的性能比定义为

最小化问题 c≥c*　　最大化问题，c*≥c

$$\gamma = \max\left\{\frac{c}{c^*}, \frac{c^*}{c}\right\}$$

γ ＞1；且γ越大，近似解越差！

- 在通常情况下，该性能比是问题输入规模n的一个函数ρ(n)，即

$$\max\left\{\frac{c}{c^*}, \frac{c^*}{c}\right\} \le \rho(n)$$

近似算法的精度

- 对于一个对某些函数 f 最小化的问题来说，可以用近似解的**相对误差**规模

$$r(s_a) = \left| \frac{f(s_a) - f(s^*)}{f(s*)} \right|$$

来度量它的精度，其中$s*$ 是问题的一个精确解。

- 另一种度量方法是，因为 $r(s_a) = \left| \frac{f(s_a)}{f(s*)} - 1 \right|$, 可以简单地使用

性能比 $r(s_a) = \max\left(\frac{f(s_a)}{f(s*)}, \frac{f(s*)}{f(s_a)} \right)$ 作为s_a的精确度量。

- $r(s_a)$越接近1，算法近似解的质量就越高 。

近似算法的性能比

- 对于问题的所有实例，它们可能的$r(s_a)$的上界，被称为该算法的**性能比**，计作R_A。
- 性能比是一个用来指出近似算法质量的主要指标，我们需要那些R_A尽量接近1的近似算法。
- 如果一个多项式时间的近似算法的性能比是c的话，我们也把它称作一个c近似算法；

近似算法应用举例

1. 顶点覆盖问题
2. 售货员旅行问题
3. 集合覆盖问题

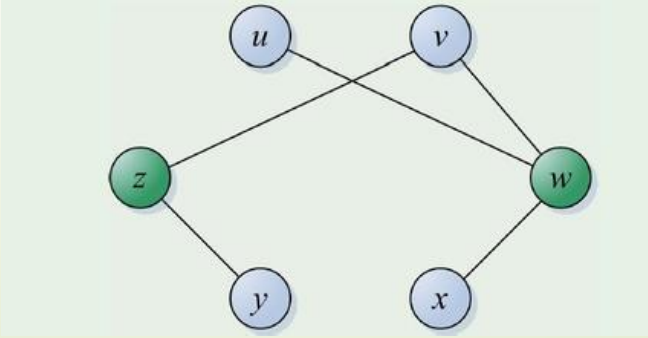
1、顶点覆盖问题的近似算法

定义1 设图$G = (V, E)$, $V' \subseteq V$ 如果图G的每条边都至少有一个顶点在V'中,则称V'是G的一个顶点覆盖。

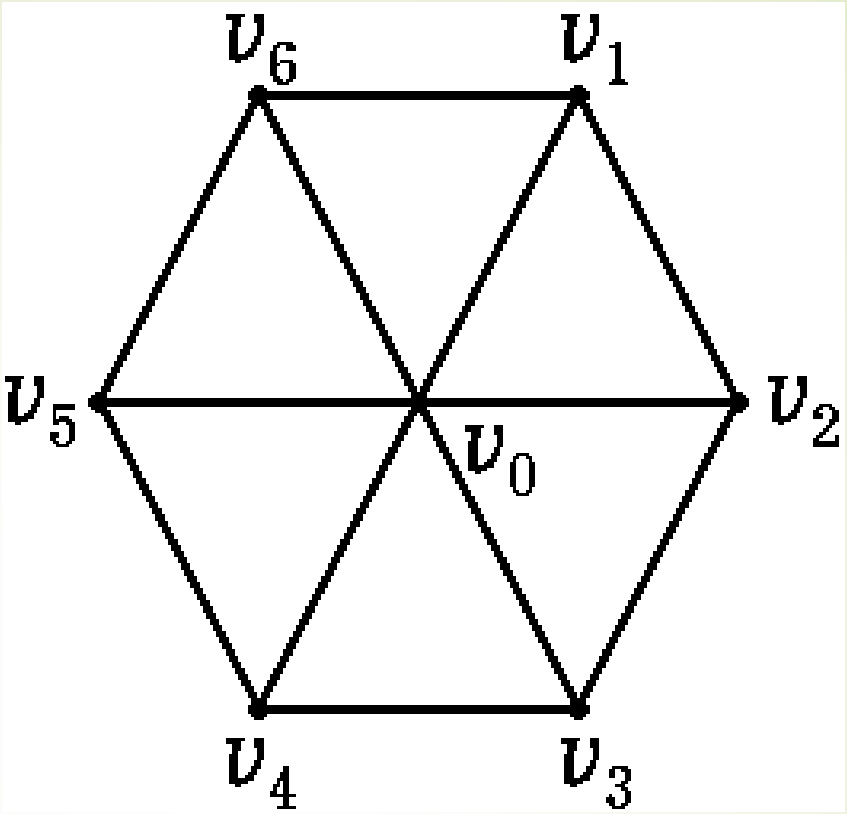
若G的一个顶点覆盖中任意去掉一个点后不再是顶点覆盖, 则称此顶点覆盖是G的一个极小顶点覆盖。

顶点数最少的点覆盖, 称为G的最小顶点覆盖。

V'的大小是它所包含的顶点个数$|V'|$



例

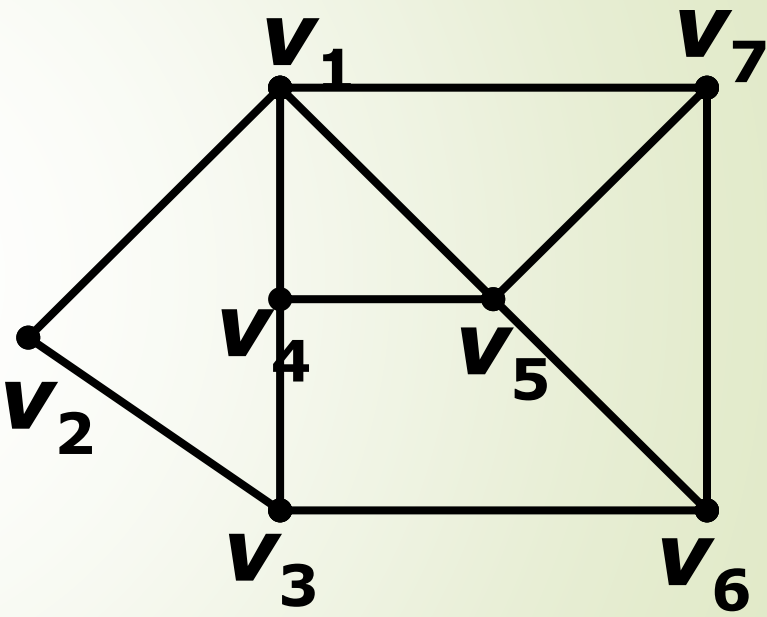


$\{v_0, v_2, v_3, v_5, v_6\}$是极小顶点覆盖。
$\{v_0, v_1, v_3, v_5\}$,$\{v_0, v_2, v_4, v_6\}$都是最小顶点覆盖。

系统监控问题：

假设v1, v2, … , v7是7个哨所,监视着11条路段(如下图所示),为节省人力,问至少需要在几个哨所派人站岗，就可以监视全部路段?

即求最小顶点覆盖问题。



{v1, v3, v5, v6}和{v1, v3, v5, v7}都是最小点覆盖, 所以至少需要在4个哨所派人站岗来监视全部路段。

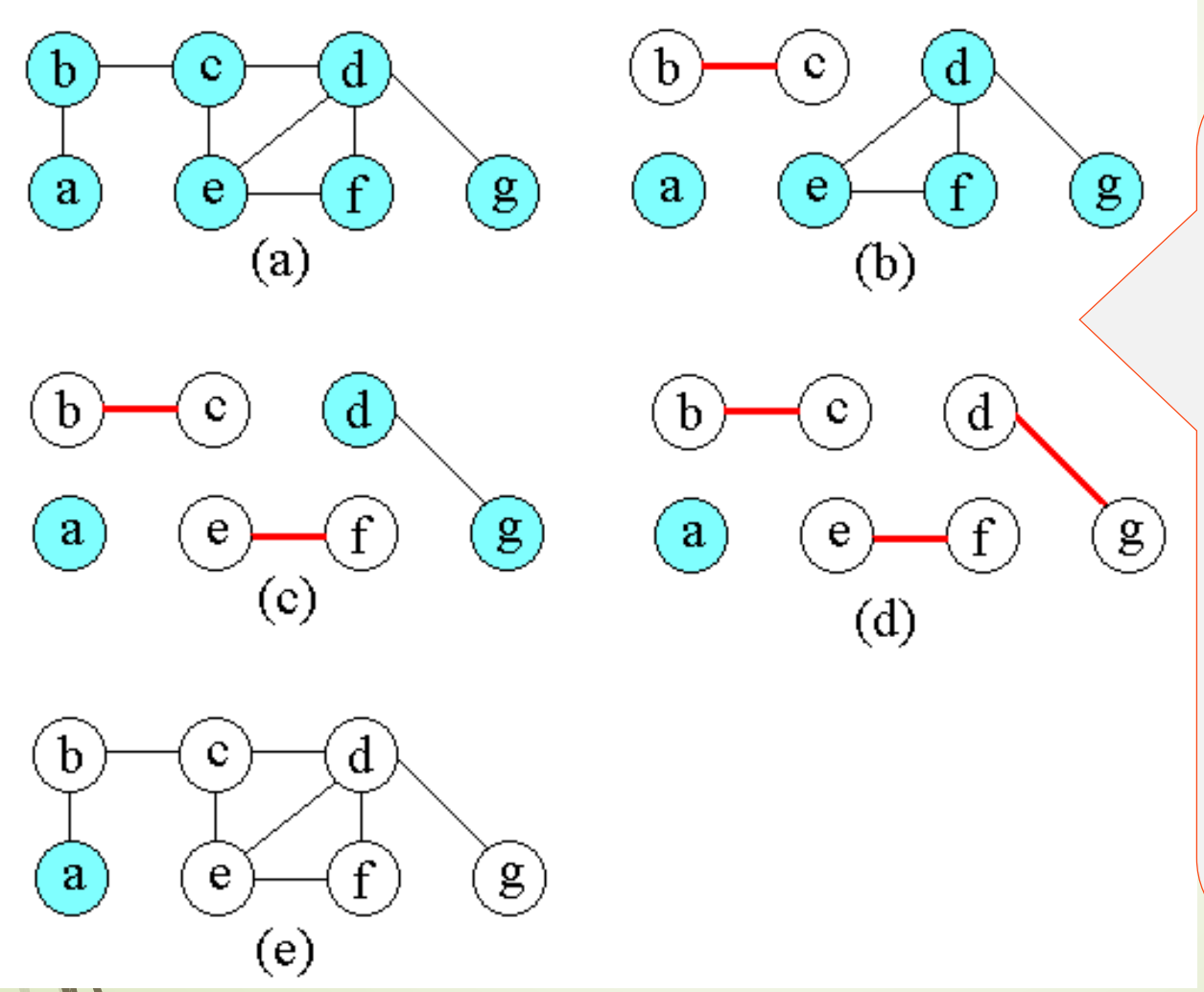
到目前为止, 还没有找到求最小顶点覆盖的多项式时间算法(算法步数不超过nc, n为G的顶点数, c为常数)。

顶点覆盖问题的近似算法

cset用来存储顶点覆盖中的各顶点。初始为空，不断从边集e1中选取一边(u,v)，将边的端点加入cset中，并将e1中已被u和v覆盖的边删去，直至cset已覆盖所有边。即e1为空。

```
VertexSet approxVertexCover ( Graph g )
{
    cset=∅;
    e1=g.e;
    while (e1 != ∅) {
      从e1中任取一条边(u,v);
      cset=cset∪{u,v};
      从e1中删去与u和v相关联的所有边;
    }
    return c
}
```

顶点覆盖问题的例子



图(a)～(e)说明了算法的运行过程及结果。(e)表示算法产生的近似最优顶点覆盖cset，它由顶点b,c,d,e,f,g所组成。(f)是图G的一个最小顶点覆盖，它只含有3个顶点：b,d和e。

顶点覆盖问题的例子　绝对近似

近似算法approxVertexCover的性能分析：

最差情况：用A存储算法循环中选取的边的集合，A中任何两条边没有公共端点，算法终止时有 |cset|=2|A|。

图G的任一顶点覆盖一定包含A中各边的至少一个端顶点，则最小顶点覆盖 |cset*|≥|A|，由此可得：

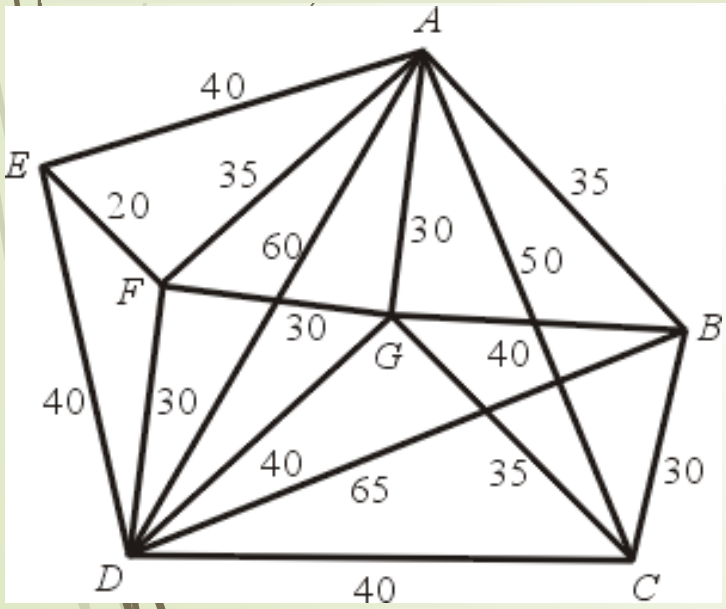
$$|cset| \leqslant 2 \,|\, cset^* \,|$$

算法approxVertexCover的性能比为2

2、Traveling Salesman Problem（TSP）售货员旅行问题近似算法

NP 问题的代表问题之一是售货员旅行问题 (traveling salesman problem)。

有一个售货员要开汽车到 n 个指定的城市去推销貨物，他必须经过全部的 n个城。現在他有一个n 城的地图及各城之间的公路距离，试问他应如何取最短的行程从家中出发再到家中？



A,*B*,*C*,…,*G* 表示 7 個城市，而售貨員要從 *A* 城出發再回到 *A* 城并访问 *B*,*C*,…,*G*所有的城，一個可行的方法是

A->B->C->D->E->F->G->A　第一路径总长235

這是否是最短的途徑？也許

A->B->C->G->D->F->E->A　第二路径总长230

有7个城，共有 6!=720 个排法。如有20个城，则有19！种排法。 $19! \simeq 1.21 \times 10^{17}$

每秒排一次，要排 3.84×10^{9} 年（一年约$3.15*10^{7}$ 秒）

2、Traveling Salesman Problem（TSP）售货员旅行问题近似算法

问题描述：给定一个完全无向图G=(V,E)，其每一边(u,v)∈E有一非负整数费用c(u,v)。要找出G的最小费用哈密顿回路。

Nearest-neighbor algorithm

最邻近算法（一种直观推断的近似算法）：

1 任意选择一个城市作为开始。

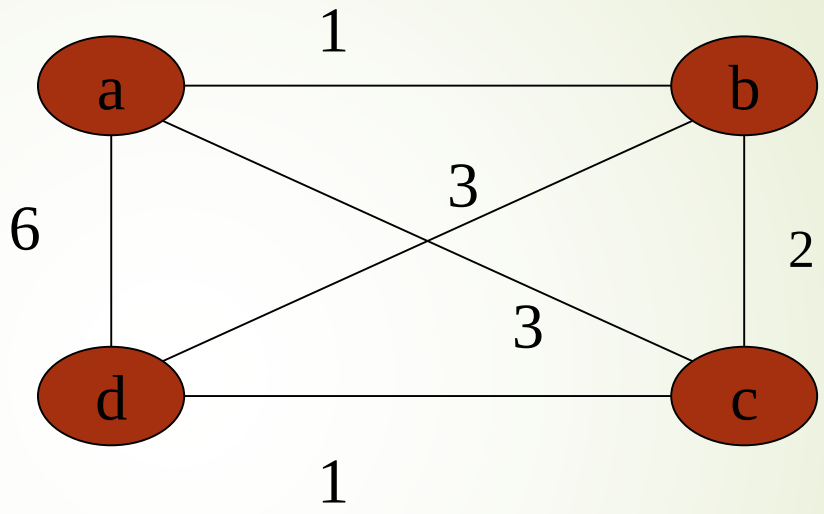
2 重复以下操作直到访问完所有的城市：

访问和最近一次访问的城市k最接近的未访问城市（顺序是无关紧要的）。

3 回到开始城市。

售货员旅行问题近似算法的性能比

实例：



最邻近算法：Sa: a-b-c-d-a 10

最优解：S*: a-b-d-c-a 8

$P(n)= Sa/ S^* =10/8 =1.25$

最近邻算法的性能比为1. 25

售货员旅行问题近似算法的性能比

- 除了简单之外，最近邻居算法乏善可陈：

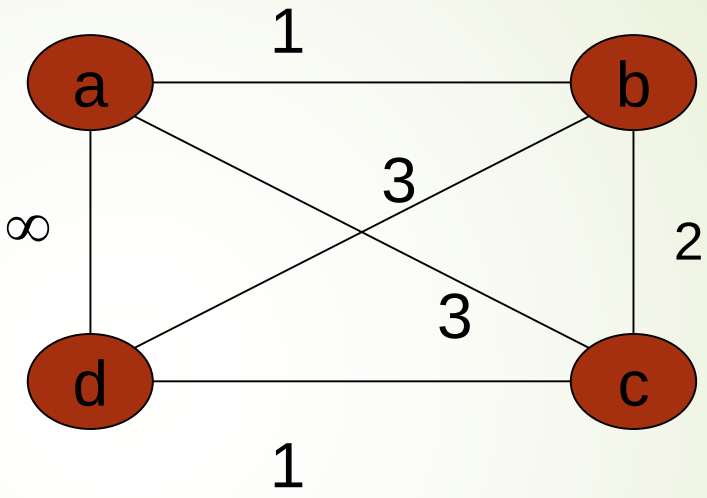
  无法给出在一般情况下，该算法求解的精确度是多少。

  为什么？

  因为该算法会在旅程别无选择的情况下，迫使我们穿过一条非常长的边。

售货员旅行问题近似算法

实例：



最邻近算法：Sa: a-b-c-d-a 4+w,(a,d)=w, w= ∞

最优解：S*: a-b-d-c-a 8

P(n)= Sa/ S* =(4+w)/8 =∞

最近邻算法的性能比为∞

近似算法

- 要求对问题的所有实例成立
- 不是总能够给出绝对差界或性能比

满足三角不等式的售货员旅行问题 欧几里德类型实例

对于一类非常重要的，被称为**欧几里德类型**的实例所构成的子集，我们可以给出最近邻居算法精确度的一个有效断言。

特殊的售货员旅行问题：

- 费用函数c往往具有三角不等式性质，即对任意的3个顶点u,v,w∈V，有：$c(u,w) \le c(u,v)+c(v,w)$。
- 当图G中的顶点就是平面上的点，任意2顶点间的费用就是这2点间的欧氏距离时，费用函数c就具有三角不等式性质。
- 对称性：d[i,j]=d[j,i]

Twice-around-the-tree algorithm
绕树两周算法

- 该算法利用了同一个图的哈密顿回路和生成树之间的联系。

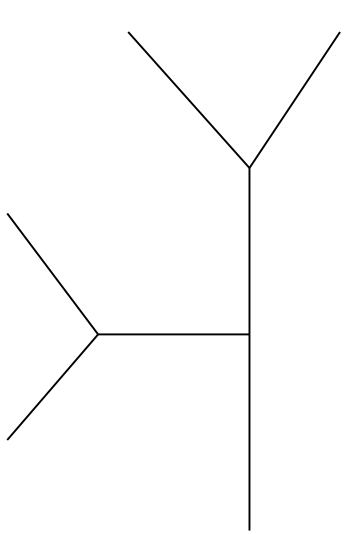
  第一步：对一个TSP的给定实例，构造它相应图的最小生成树。

  第二步：从一个任意顶点开始，绕着这棵最小生成树散步一周，并记录下经过的顶点。

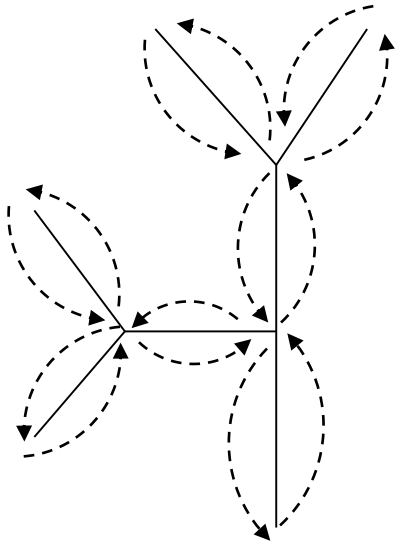
  第三步：扫描第二步中得到的顶点列表，从中消去所有重复出现的顶点，但留下出现在列表尾部的起始顶点。

列表中余下的顶点就构成了一条哈密顿回路，这就是该算法的输出。

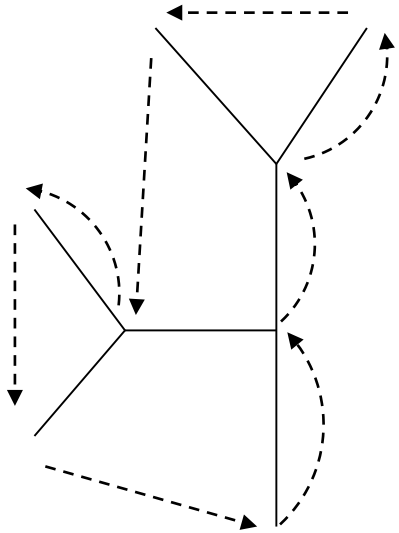
Twice-around-the-tree algorithm
绕树两周算法



(a)
A spanning tree



(b)
Twice around the tree



(c)
A tour with shortcut

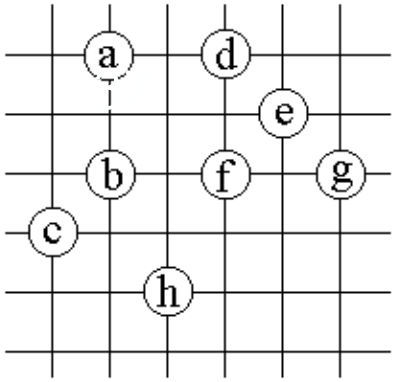
Twice-around-the-tree algorithm
绕树两周算法

void approxTSP (Graph g)
{

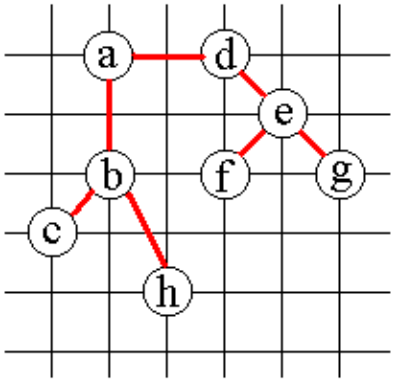
1. 选择g的任一顶点r；
2. 用Prim算法找出带权图g的一棵以r为根的最小生成树T；
3. 前序遍历树T得到的顶点表L；
4. 将r加到表L的末尾，按表L中顶点次序，去掉重复顶点，组成回路H，作为计算结果返回；

}

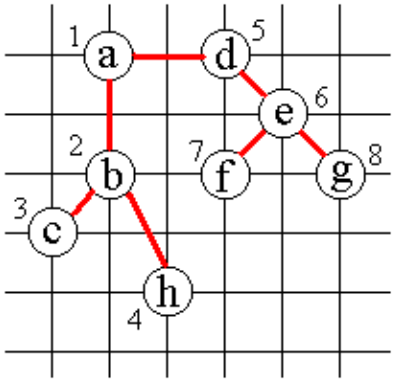
算法的时间复杂性：$O(n^2)$



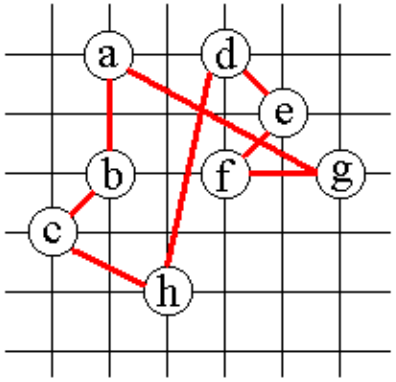
(a)



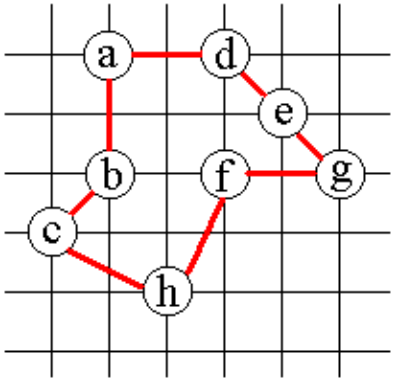
(b)



(c)



(d)



(e)

线路：

a→b→c→b→h→b→a→d→e→f→e→g→e→d→a

L=(a, b, c, h, d, e, f, g)

H=(a→b→c→h→d→e→f→g→a)

H*=(a→b→c→h→f→g→e→d→a)

(b)表示找到的最小生成树T；(c)表示对T作前序遍历的次序；(d)表示L产生的哈密顿回路H；(e)是G的一个最小费用旅行售货员回路。

绕树两周算法的性能比

当费用函数满足三角不等式时，算法找出的旅行售货员回路的费用不会超过最优旅行售货员回路费用的2倍。

- 绕树两周是一个多项式时间的算法；
- 需要证明，对于旅行商问题的欧几里德实例来说，绕树两周算法得到的旅程s_a的长度最多为最优旅程s^*长度的两倍；即，

$$f(s_a) \leq 2\, f(s^*)$$

一般的旅行售货员问题

在费用函数不一定满足三角不等式的一般情况下，不存在具有常数性能比的解TSP问题的多项式时间近似算法，除非**P=NP**。换句话说，若P≠NP，则对任意常数ρ>1，不存在性能比为ρ的解旅行售货员问题的多项式时间近似算法。

$$c \leq \rho c^*$$

3、集合覆盖问题的近似算法

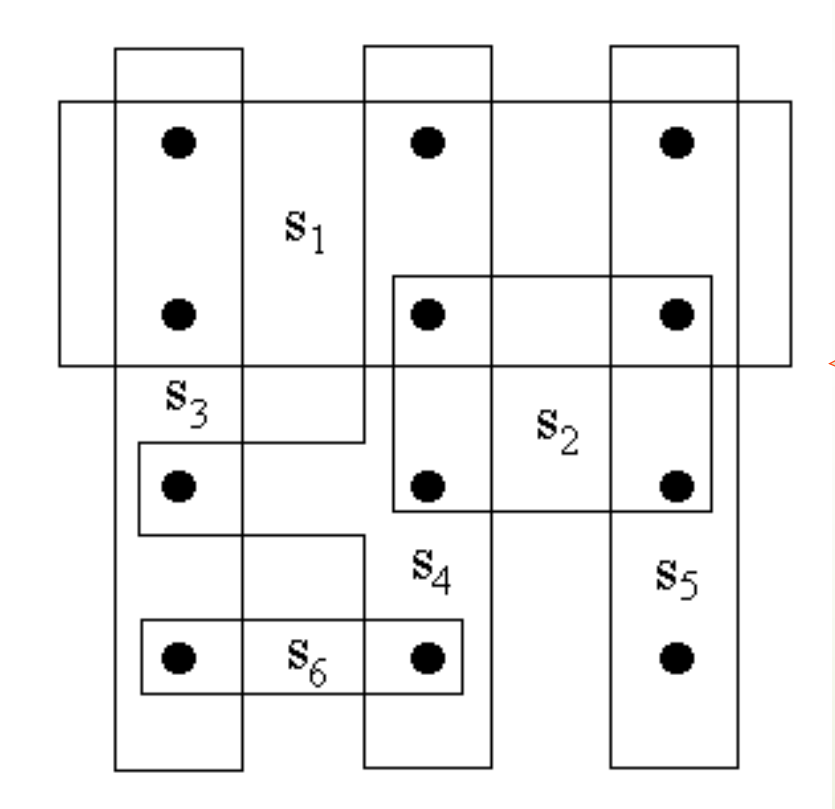
集合覆盖问题是一个最优化问题。

集合覆盖问题的一个实例〈X,F〉由一个有限集X及 X 的一个子集族 F 组成。子集族 F 覆盖了有限集X。也就是说 X中每一元素至少属于F中的一个子集。对于F 中的一个子集C∈F，若C中的X的子集覆盖了X，则称C覆盖了X。集合覆盖问题就是要找出F中覆盖X的最小子集C^*，使得

$$|C^*|=\min\{\,|C| \mid C\subseteq F \text{ 且C覆盖X}\,\}$$

3　集合覆盖问题的近似算法

集合覆盖问题举例：



用12个黑点表示集合X。
F={S1,S2,S3,S4,S5,S6,}，如图所示。
容易看出，对于这个例子，最小集合覆盖为：
C={S3,S4,S5}。

3　集合覆盖问题的暴力解法

1. 取出{S1, S2, ..., Sn}的所有子集, 共$O(2^n)$种情况
2. 计算该子集内部是否有两个元素有交集, 这是一个双重循环, 时间复杂度$O(n^2)$

暴力法的时间复杂度为$O(2^n * n^2)$

3　集合覆盖问题的近似算法

集合覆盖问题近似算法——贪心算法

```
Set greedySetCover (X,F)
{
    U=X;
    C=∅;
    while (U !=∅) {
      选择F中使|S∩U|最大的子集S;
      U=U-S;
      C=C∪{S};
      }
    return C;
}
```

算法的循环体最多执行min{|X|，|F|}次。而循环体内的计算显然可在O(|X||F|)时间内完成。因此，算法的计算时间为O(|X||F|min{|X|，|F|})。由此即知，该算法是一个多项式时间算法。

解：c={s1,s4,s5,s6}

END