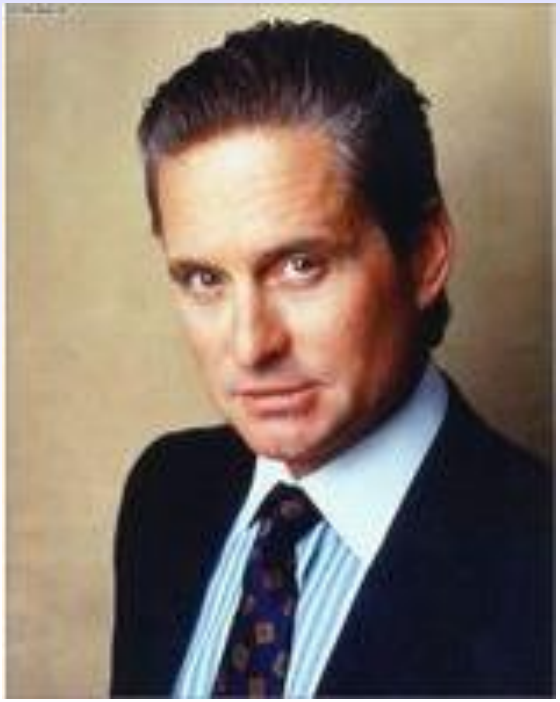
贪心法

贪婪，我找不到一个更好的词来描述它，它就是好！它就是对！它就是有效！

———美国演员道格拉思，在影片《华尔街》中的台词



影视作品

《华尔街2，金钱永不眠》 (2010)
《完美的谋杀》Perfect Murder (1998)
《心理游戏》Game, The (1997)
《美国总统》American President (1995)
《本能》Basic Instinct (1992)
《黑雨》Black Rain (1989)
《玫瑰战争》War of the Roses (1989)
《华尔街》Wall Street (1987)
《高度怀疑》Beyond a reasonable doubt(2009)
《致命诱惑》Fatal Attraction (1987)

思考

- 你的朋友要开一家安全公司，这需要得到n个不同的密码软件的许可证．根据规章，他们只能以每个月至多一个的速度得到这些许可证。每个许可证目前的售价是100美元，但他们全都按指数增长曲线变得更贵，设许可证j的费用每个月按照$r_j>1$的因子增加，且假设所有价格增长率是不同的，应该按照什么次序买这些许可证使花的总钱数最少？

思考题

- 让我们考虑一条长的乡村道路，沿着它非常稀疏地分布着一些房子。不管这种田园诗般的环境，假设所有这些房子的居民都是热心的蜂窝电话用户。你想把蜂窝电话基站放在沿着这条路的某些点上，使得每个房子都在一个基站的4英里之内。
- 给出一个有效算法，使用尽可能少的基站来实现这个目标。

什么是贪婪

- “良禽择木而栖，良将选主而事” 。人类似乎永远在追求最好的东西。用到算法设计中，就是贪婪选择算法或贪婪算法。

- 什么是贪婪算法呢？

- 每步只看眼前效果最好的做法就是贪婪，以贪婪作为决策基础的算法就是贪婪算法。

- 贪婪算法的基本策略：一步一步地构建问题的最优解决方案，其中每一步只需考虑眼前的最佳选择（局部判断规则），即通过局部最优到达全局最优。

贪心算法（1）

- **找零钱1**

  假设有4种硬币，它们的面值分别为二角五分、一角、五分、一分。现在要给顾客找零六角三分钱，怎么找使得硬币数目最少？

通过作出在当前看来最优的决策（贪心选择），将原问题规模缩小，如此反复，直至得到最终解–贪心算法

贪心算法（2）

- **找零钱2**

  假设有3种硬币，它们的面值分别为一角一分、五分、一分。现在要找给顾客一角五分钱，怎么找使得硬币数目最少？

贪心算法并非对所有问题都能得到整体最优解！！

贪心算法（3）

贪心算法总是作出在当前看来最好的决策。也就是说贪心算法**并不从整体最优考虑**，它所作出的决策只是在某种意义上的**局部最优**决策。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**。

贪心法的设计思想

贪心法通过一系列步骤构造完整解来解决问题。

只根据当前已有的就做出选择，而且一旦做出了选择，这个选择都不会改变。

换言之，贪心法并不是从整体最优考虑，它所做出的选择只是在某种意义上的局部最优。

这种局部最优选择并不总能获得整体最优解（**Optimal Solution**），但通常能获得近似最优解（**Near-Optimal Solution**）。

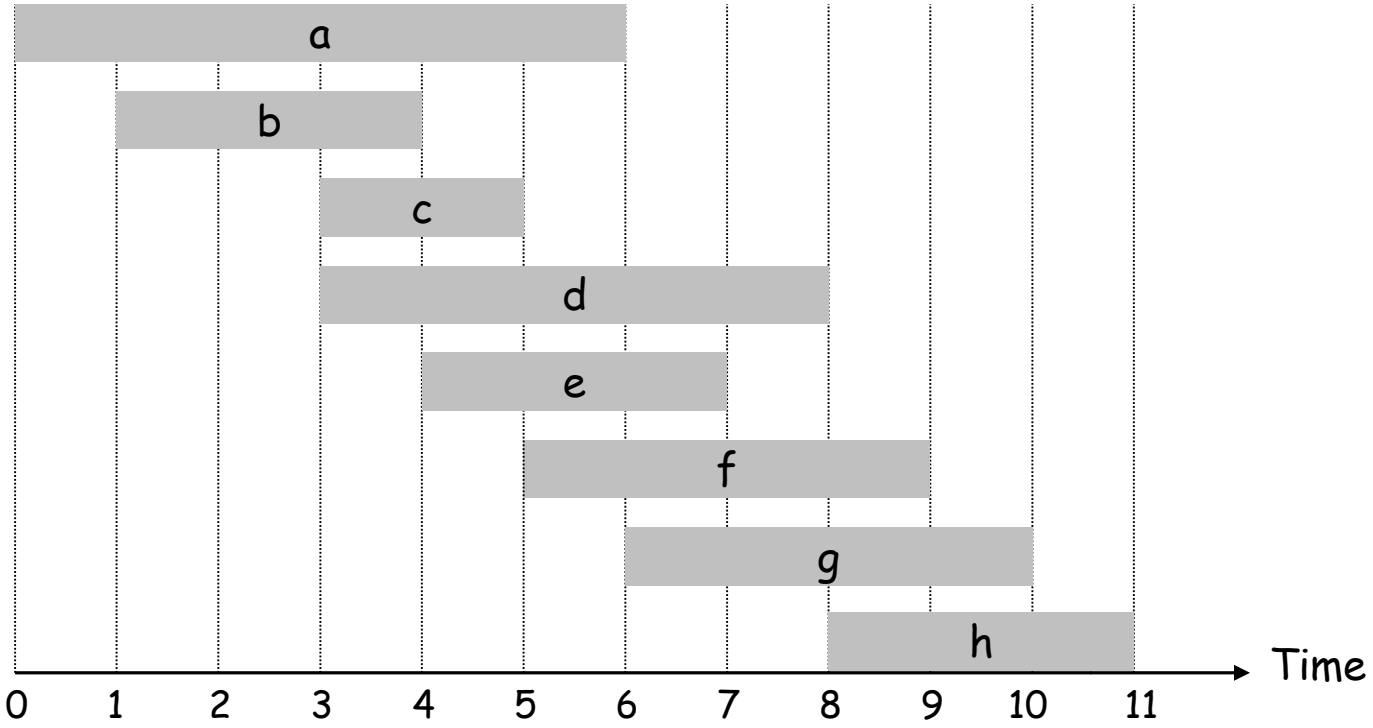
Canon EOS 450D [EF70-200mm f/4L IS USM] F4.0 1/500s ISO100

Interval Scheduling （区间调度）

Interval Scheduling

Interval scheduling.

- Job j starts at s_j（开始时间） and finishes at f_j（结束时间）.
- Two jobs compatible（相容） if they don't overlap（重叠）.
- Goal: find maximum subset of mutually compatible （互相兼容）jobs.



Interval Scheduling: Greedy（贪心） Algorithms

Greedy template. Consider jobs in some order. Take each job provided it's compatible with the ones already taken.

- [Earliest start time] （最早开始时间）
  - Consider jobs in ascending order （升序） of start time s_j.

breaks earliest start time

Interval Scheduling: Greedy Algorithms

Greedy template. Consider jobs in some order. Take each job provided it's compatible with the ones already taken.

- [Shortest interval] （最短区间长度）
  - Consider jobs in ascending order of interval length $f_j - s_j$.

breaks shortest interval

Interval Scheduling: Greedy Algorithms

Greedy template. Consider jobs in some order. Take each job provided it's compatible with the ones already taken.

- [Fewest conflicts]（最小冲突区间）
  - For each job, count the number of conflicting jobs c_j. Schedule in ascending order of conflicts c_j.

breaks fewest conflicts

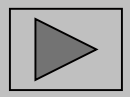
Interval Scheduling: Greedy Algorithms

Greedy template. Consider jobs in some order. Take each job provided it's compatible with the ones already taken.

- [Earliest finish time] （最早结束时间）
  - Consider jobs in ascending order of finish time f_j.

```
Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

A ← φ
for j = 1 to n {
   if (job j compatible with A)
      A ← A ∪ {j}
}
return A
```



Implementation. O(n log n).

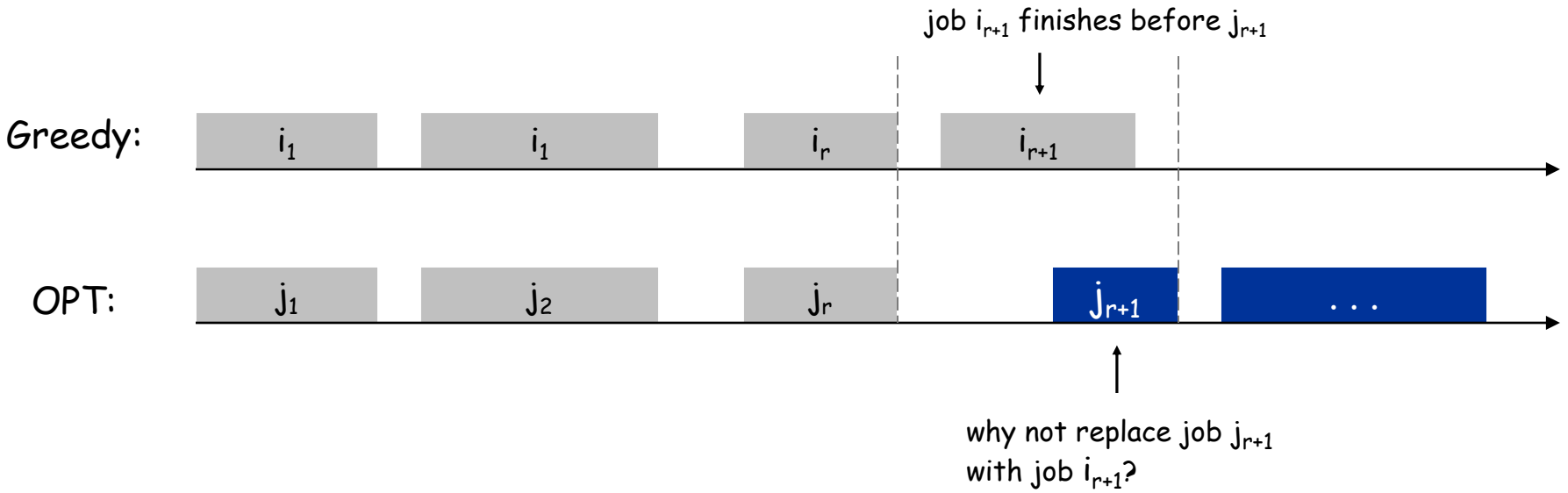
Remember job j* that was added last to A. Job j is compatible with A if $s_j \geq f_{j^*}$.

Interval Scheduling: Analysis

Theorem. Greedy algorithm is optimal（最优的）.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let $i_1, i_2, \ldots i_k$ denote set of jobs selected by greedy.
- Let $j_1, j_2, \ldots j_m$ denote set of jobs in the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of r.

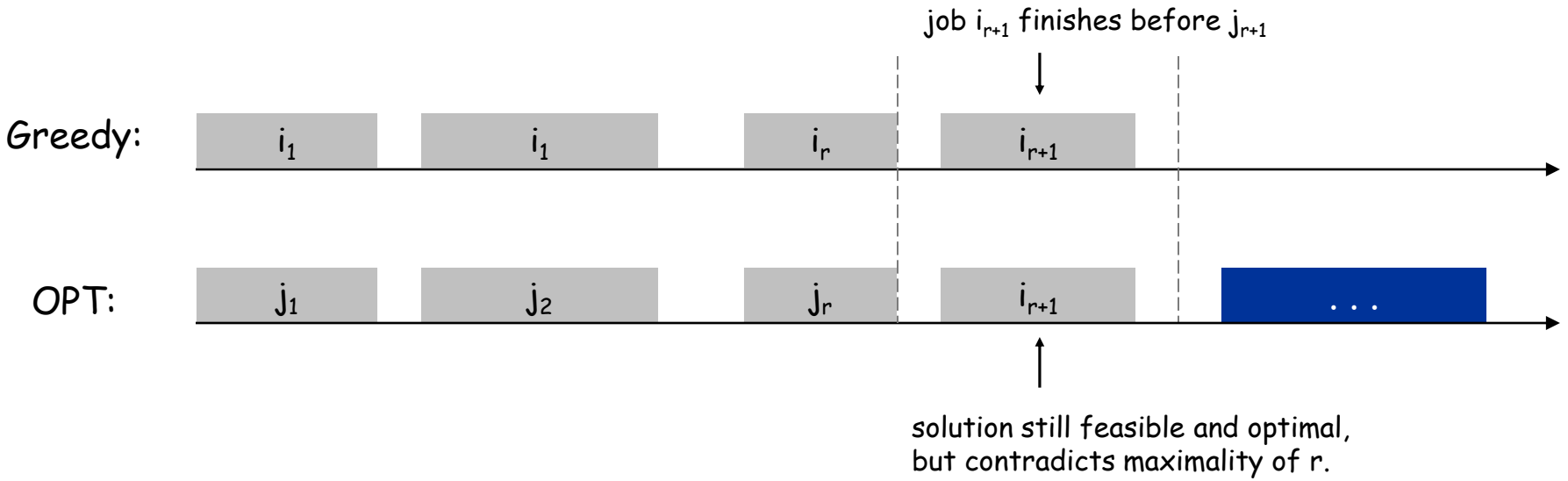


Interval Scheduling: Analysis

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let $i_1, i_2, \ldots i_k$ denote set of jobs selected by greedy.
- Let $j_1, j_2, \ldots j_m$ denote set of jobs in the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of r.



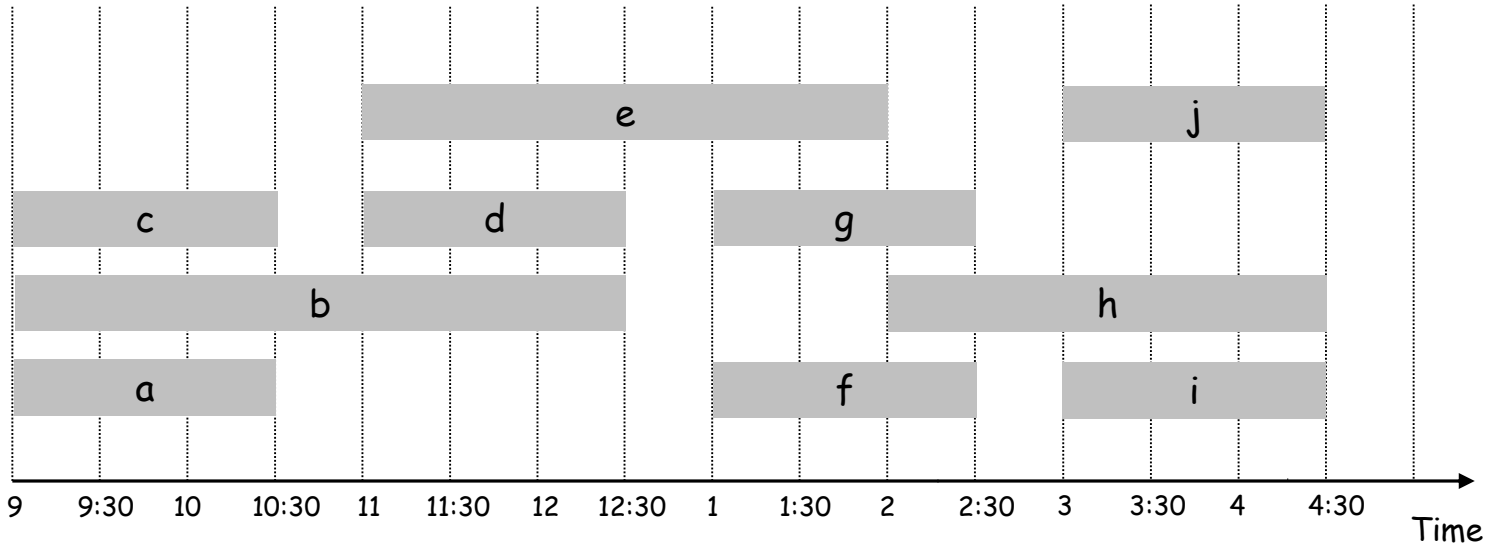
4.1 Interval Partitioning（区间划分）

Interval Partitioning （区间划分）

Interval partitioning.

- Lecture j starts at s_j and finishes at f_j.
- Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

Ex: This schedule uses 4 classrooms to schedule 10 lectures.

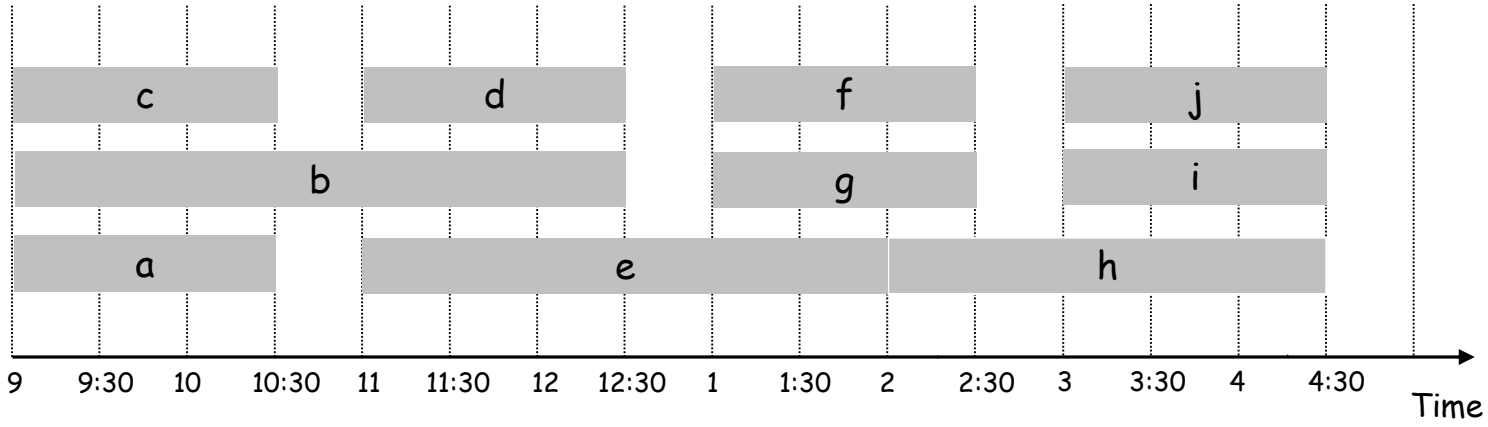


Interval Partitioning

Interval partitioning.

- Lecture j starts at s_j and finishes at f_j.
- Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

Ex: This schedule uses only 3.



Interval Partitioning: Lower Bound on Optimal Solution

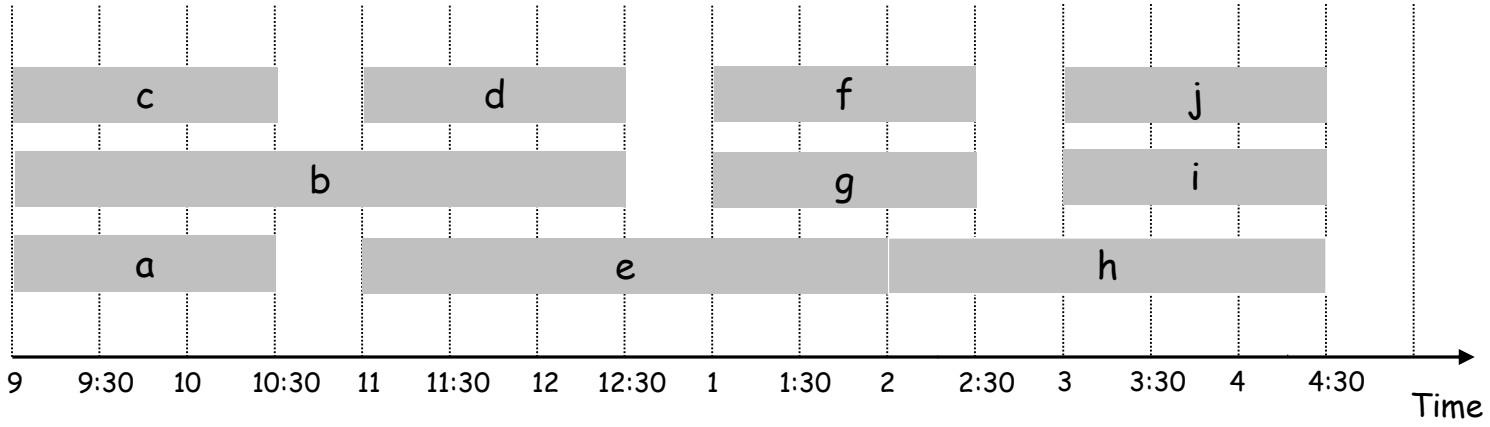
Def. The depth of a set of open intervals （区间集合深度） is the maximum number that contain any given time.

Key observation. Number of classrooms needed ≥ depth.

Ex: Depth of schedule below = 3 ⇒ schedule below is optimal.

a, b, c all contain 9:30

Q. Does there always exist a schedule equal to depth of intervals ?



Interval Partitioning: Greedy Algorithm

Greedy algorithm. Consider lectures in increasing order of start time: assign lecture to any compatible classroom.

```
Sort intervals by starting time so that s_1 ≤ s_2 ≤ ... ≤ s_n.
d ← 0   ← number of allocated classrooms

for j = 1 to n {
   if (lecture j is compatible with some classroom k)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d ← d + 1
}
```

Implementation. O(n log n).

- For each classroom k, maintain the finish time of the last job added.
- Keep the classrooms in a priority queue.

Interval Partitioning: Greedy Analysis

Observation. Greedy algorithm never schedules two incompatible lectures in the same classroom.

Theorem. Greedy algorithm is optimal.
Pf. （略）

Scheduling to Minimize Lateness

（最小延迟调度）

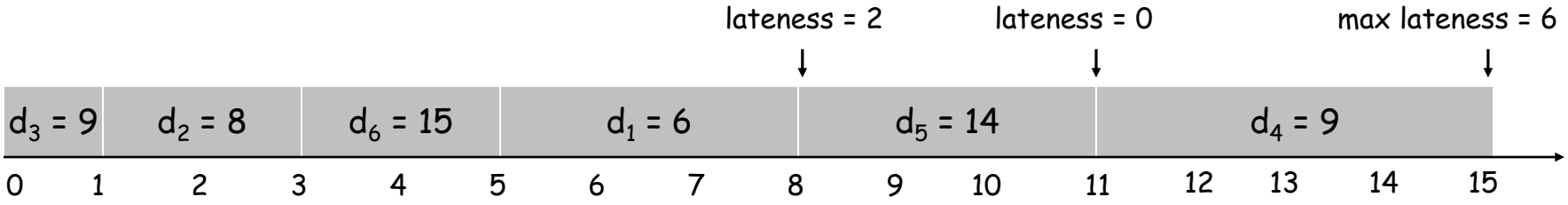
Scheduling to Minimizing Lateness （最小延迟调度）

Minimizing lateness problem.

- Single resource processes one job at a time.
- Job j requires t_j units of processing time and is due at time d_j.
- If j starts at time s_j, it finishes at time $f_j = s_j + t_j$.
- Lateness: $\ell_j = \max \{ 0, f_j - d_j \}$.
- Goal: schedule all jobs to minimize maximum lateness L = $\max \ell_j$.

Ex:

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time t_j.

- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

- [Earliest deadline first] Consider jobs in ascending order of deadline d_j.

Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time t_j.

	1	2
t_j	1	10
d_j	100	10

counterexample

- [Smallest slack （松弛）] Consider jobs in ascending order of slack $d_j - t_j$.

	1	2
t_j	10	1
d_j	10	2

counterexample

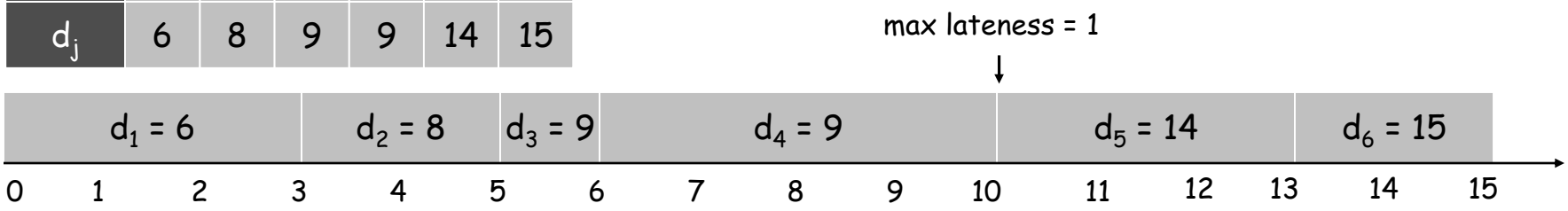
Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

```
Sort n jobs by deadline so that d_1 ≤ d_2 ≤ … ≤ d_n

t ← 0
for j = 1 to n
   Assign job j to interval [t, t + t_j]
   s_j ← t, f_j ← t + t_j
   t ← t + t_j
output intervals [s_j, f_j]
```

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



贪婪策略要想获得最优解，必须满足如下条件：

（1）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，也称此问题满足最优性原理。

（2）贪心选择性质

所谓贪心选择性质是指问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来得到。

贪婪策略也是一种分治策略，与一般分治不同的是，贪婪策略每步解决的子问题数量为1个。

如果一个问题不具备上述条件，并不说明不能采用贪婪策略，只不过贪婪策略将不能保证获得最优解。在面对难解问题时，贪婪策略是我们使用的法宝之一。

图问题中的贪心法

1. TSP问题
2. 图着色问题
3. 最小生成树问题

1. TSP问题

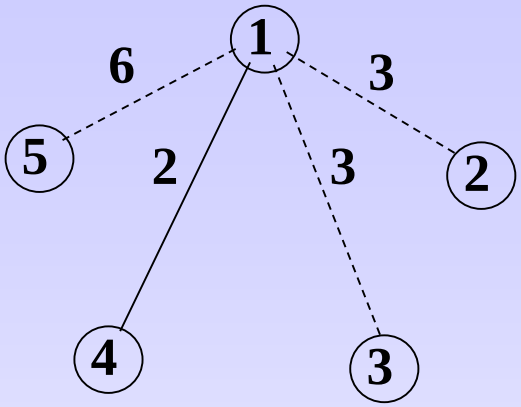
求解TSP问题至少有两种贪心策略是合理的：

（1）最近邻点策略：从任意城市出发，每次在没有到过的城市中选择最近的一个，直到经过了所有的城市，最后回到出发城市。

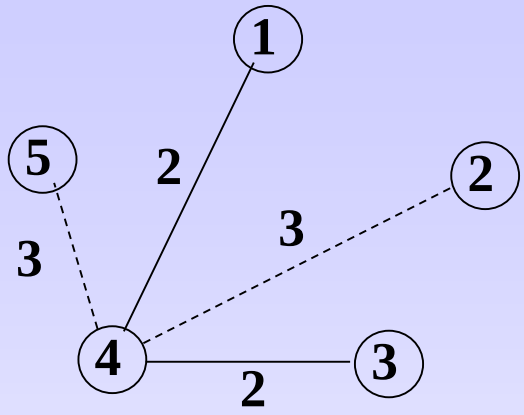
C=

$$\begin{pmatrix} \infty & 3 & 3 & 2 & 6 \\ 3 & \infty & 7 & 3 & 2 \\ 3 & 7 & \infty & 2 & 5 \\ 2 & 3 & 2 & \infty & 3 \\ 6 & 2 & 5 & 3 & \infty \end{pmatrix}$$

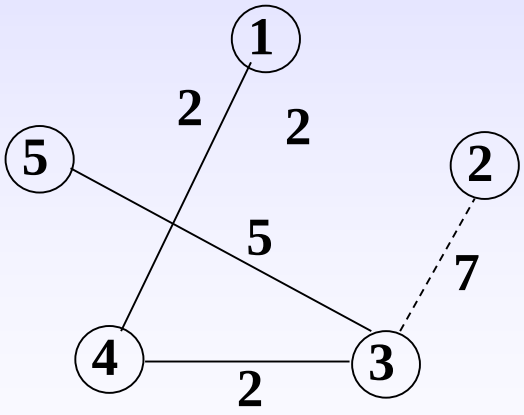
(a) 5城市的代价矩阵



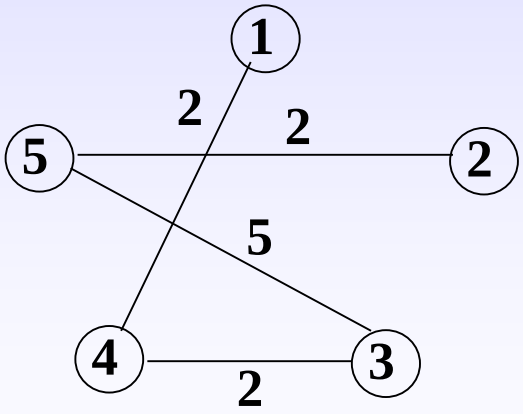
(b) 城市1→城市4



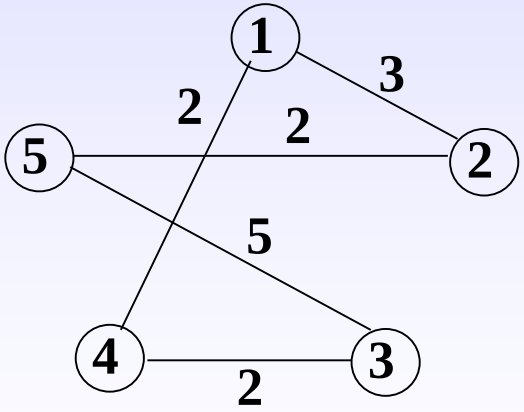
(c) 城市4→城市3



(d) 城市3→城市5



(e) 城市5→城市2



(f) 城市2→城市1

最近邻点贪心策略求解TSP问题的过程

设图G有n个顶点，边上的代价存储在二维数组w[n][n]中，集合V存储图的顶点，集合P存储经过的边，最近邻点策略求解TSP问题的算法如下：

算法7.1——最近邻点策略求解TSP问题

1. P={ };
2. V=V−{u_0}; u=u_0;　//从顶点u_0出发
3. 循环直到集合P中包含n−1条边
   - 3.1 查找与顶点u邻接的最小代价边(u, v)并且v属于集合V;
   - 3.2 P=P+{(u, v)};
   - 3.3 V=V−{v};
   - 3.4 u=v;　//从顶点v出发继续求解

算法7.1的时间性能为$O(n^2)$，因为共进行n-1次贪心选择，每一次选择都需要查找满足贪心条件的最短边。

用最近邻点贪心策略求解TSP问题所得的结果不一定是最优解，图7.1(a)中从城市1出发的最优解是1→2→5→4→3→1，总代价只有13。当图中顶点个数较多并且各边的代价值分布比较均匀时，最近邻点策略可以给出较好的近似解，不过，这个近似解以何种程度近似于最优解，却难以保证。例如，在图7.1中，如果增大边(2, 1)的代价，则总代价只好随之增加，没有选择的余地。

（2）最短链接策略：每次在整个图的范围内选择最短边加入到解集合中，但是，要保证加入解集合中的边最终形成一个哈密顿回路。因此，当从剩余边集E'中选择一条边(u, v)加入解集合S中，应满足以下条件：

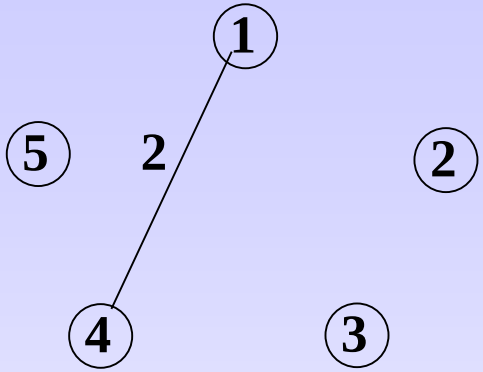
① 边(u, v)是边集E'中代价最小的边；

② 边(u, v)加入解集合S后，S中不产生回路；

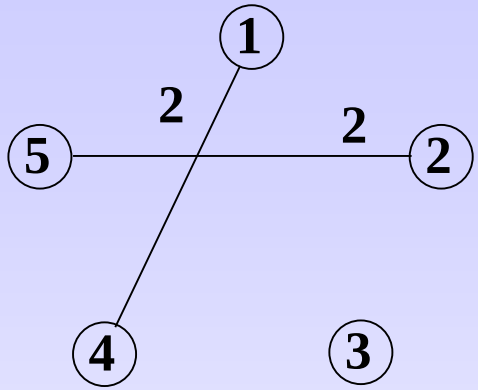
③ 边(u, v) 加入解集合S后，S中不产生分枝；

$$C=\begin{pmatrix} \infty & 3 & 3 & 2 & 6 \\ 3 & \infty & 7 & 3 & 2 \\ 3 & 7 & \infty & 2 & 5 \\ 2 & 3 & 2 & \infty & 3 \\ 6 & 2 & 5 & 3 & \infty \end{pmatrix}$$

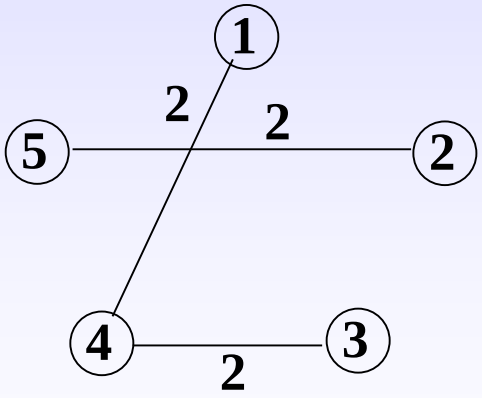
(a) 5城市的代价矩阵



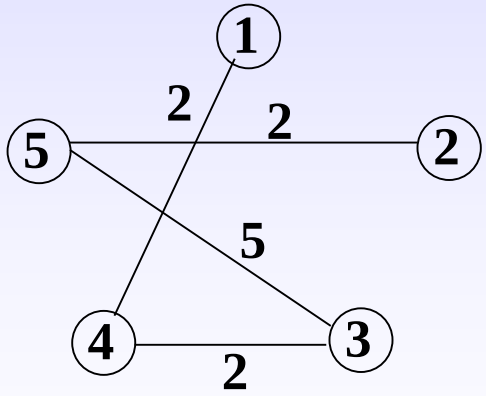
(b) 城市1→城市4



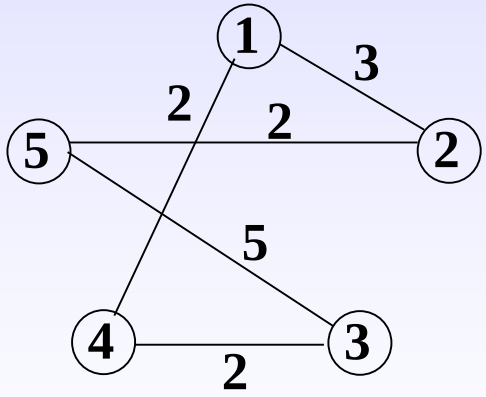
(c) 城市5→城市2



(d) 城市4→城市3



(e) 城市3→城市5



(f) 城市2回到出发城市1

最短链接贪心策略求解TSP问题的过程

设图G有n个顶点，边上的代价存储在二维数组w[n][n]中，集合E'是候选集合即存储所有未选取的边，集合P存储经过的边，最短链接策略求解TSP问题的算法如下：

算法7.2——最短链接策略求解TSP问题

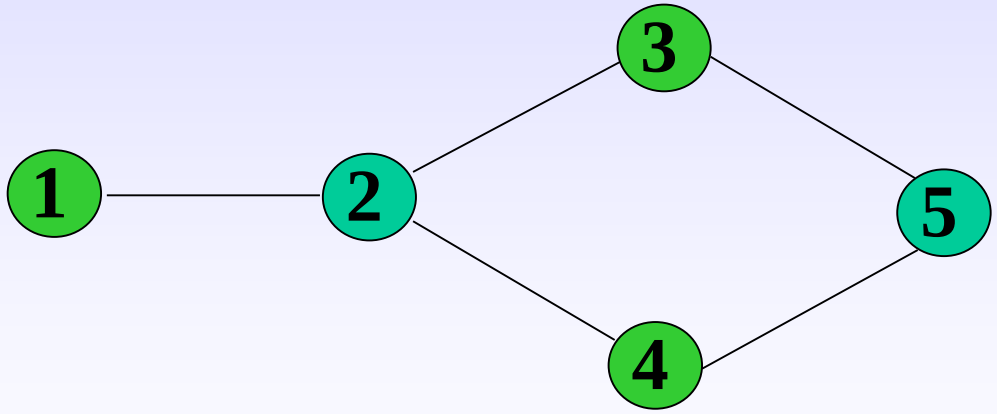
1. P={ };
2. E′=E;　//候选集合，初始时为图中所有边
3. 循环直到集合P中包含n−1条边
   3.1 在E′中选取最短边(u, v);
   3.2 E′=E′−{(u, v)};
   3.3 如果 (顶点u和v在P中不连通 and 不产生分枝)
   　　则P=P+{(u, v)};

在算法7.2中，如果操作“在E'中选取最短边(u, v)”用顺序查找，则算法7.2的时间性能是$O(n^2)$，如果采用堆排序的方法将集合E'中的边建立堆，则选取最短边的操作可以是$O(\log_2 n)$，对于两个顶点是否连通以及是否会产生分枝，可以用并查集的操作将其时间性能提高到$O(n)$，此时算法7.2的时间性能为$O(n\log_2 n)$。

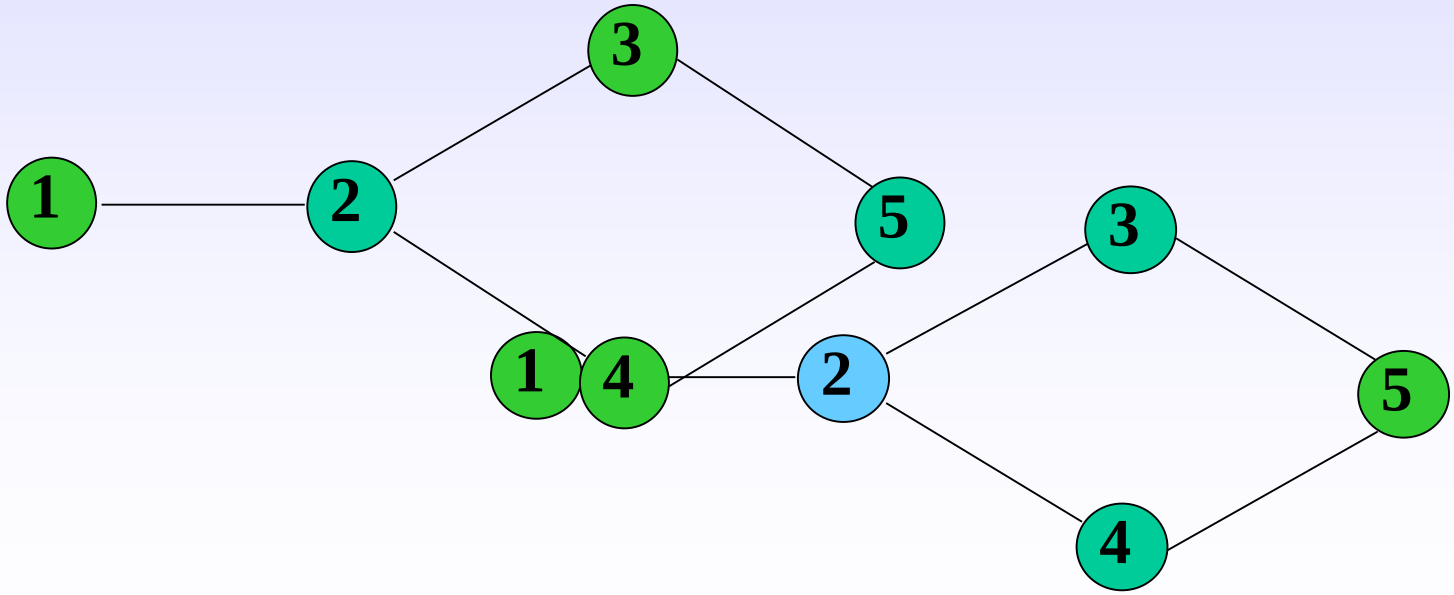
2. 图着色问题

给定无向连通图$G=(V, E)$，求图G的最小色数k，使得用k种颜色对G中的顶点着色，可使任意两个相邻顶点着色不同。

例如，图7.3(a)所示的图可以只用两种颜色着色，将顶点1、3和4着成一种颜色，将顶点2和顶点5着成另外一种颜色。为简单起见，下面假定k个颜色的集合为{颜色1, 颜色2, …, 颜色k}。



贪心策略：选择一种颜色，以任意顶点作为开始顶点，依次考察图中的未被着色的每个顶点，如果一个顶点可以用颜色1着色，换言之，该顶点的邻接点都还未被着色，则用颜色1为该顶点着色，当没有顶点能以这种颜色着色时，选择颜色2和一个未被着色的顶点作为开始顶点，用第二种颜色为尽可能多的顶点着色，如果还有未着色的顶点，则选取颜色3并为尽可能多的顶点着色，依此类推。



设数组color[n]表示顶点的着色情况，贪心法求解图着色问题的算法如下：

算法——图着色问题

1．color[1]=1; //顶点1着颜色1
2．for (i=2; i<=n; i++) //其他所有顶点置未着色状态
color[i]=0;
3．k=0;
4．循环直到所有顶点均着色
4.1 k++; //取下一个颜色
4.2 for (i=2; i<=n; i++) //用颜色k为尽量多的顶点着色
4.2.1 若顶点i已着色，则转步骤4.2，考虑下一个顶点;
4.2.2 若图中与顶点i邻接的顶点着色与顶点i着颜色k不冲突，则color[i]=k;
5．输出k;

考虑一个具有2*n*个顶点的无向图，顶点的编号从1到2*n*，当*i*是奇数时，顶点*i*与除了顶点*i*+1之外的其他所有编号为偶数的顶点邻接，当*i*是偶数时，顶点*i*与除了顶点*i*−1之外的其他所有编号为奇数的顶点邻接，这样的图称为**双向图**（Bipartite）。

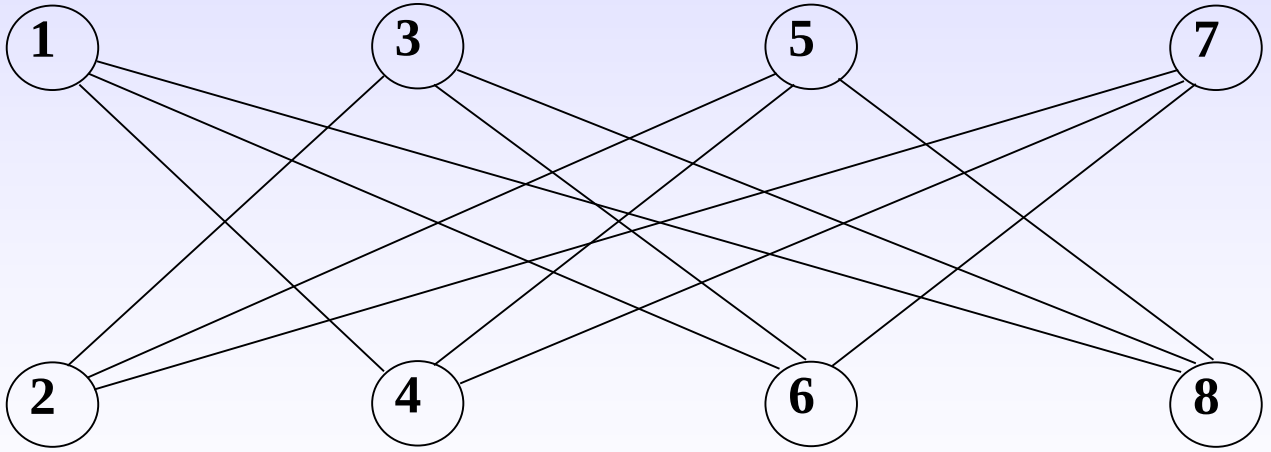


图7.4 具有8个顶点的双向图

最小生成树问题

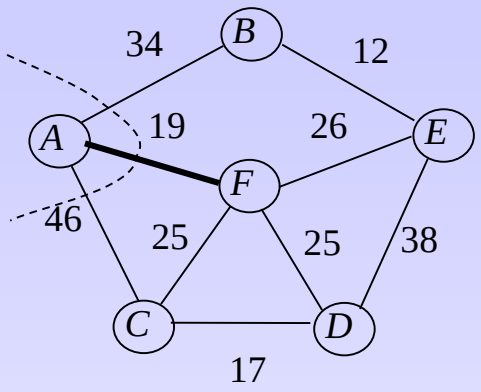
如果让你对计算机进行组网，需要在不同的计算机对之间建立连线。我们的目标是使所有的计算机连通，且连线的维护成本最低，你会怎么办？

设*G*=(*V*，*E*)是一个无向连通网，生成树上各边的权值之和称为该生成树的代价，在*G*的所有生成树中，代价最小的生成树称为最小生成树（Minimal Spanning Trees）。

最小生成树问题至少有两种合理的贪心策略：

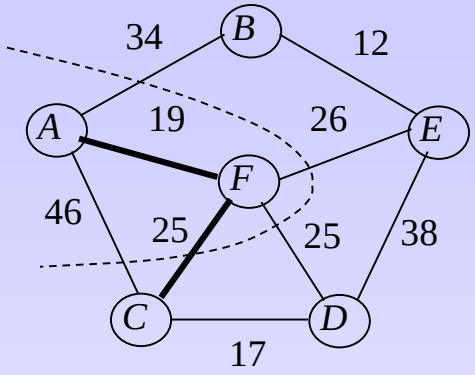
（1）最近顶点策略：任选一个顶点，并以此建立起生成树，每一步的贪心选择是简单地把不在生成树中的最近顶点添加到生成树中。

Prim算法就应用了这个贪心策略,它使生成树以一种自然的方式生长，即从任意顶点开始，每一步为这棵树添加一个分枝，直到生成树中包含全部顶点。



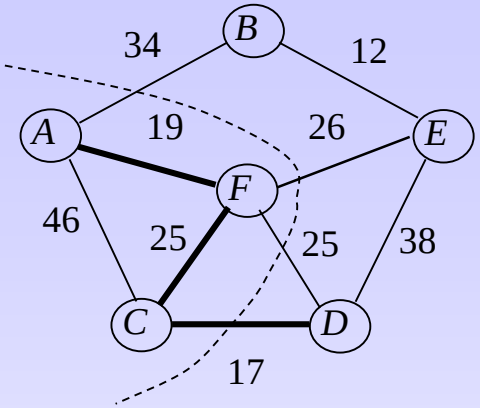
(a) 连通网，$U=\{A\}$

cost={(A, B)34,(A, C)46, (A, D)∞,(A, E)∞,**(A, F)19**}



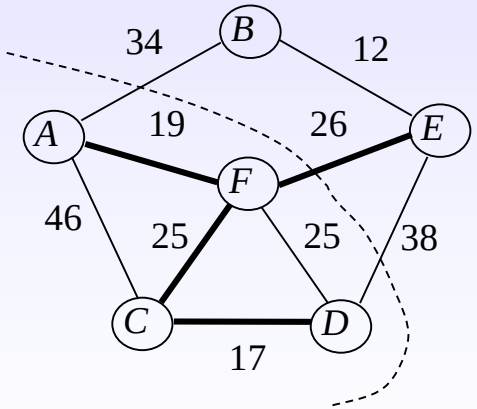
(b) $U=\{A, F\}$

cost={(A, B)34,**(F, C)25**, (F, D)25,(F, E)26}



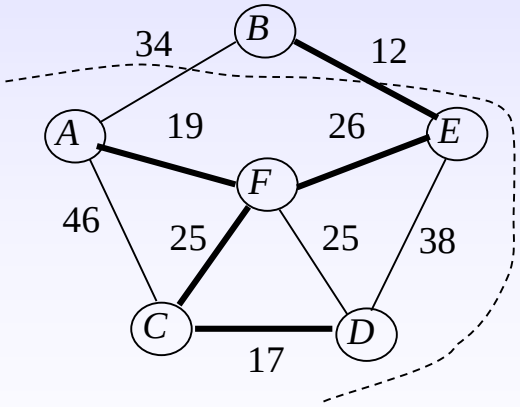
(c) $U=\{A, F, C\}$

cost={(A, B)34, **(C, D)17**,(F, E)26}



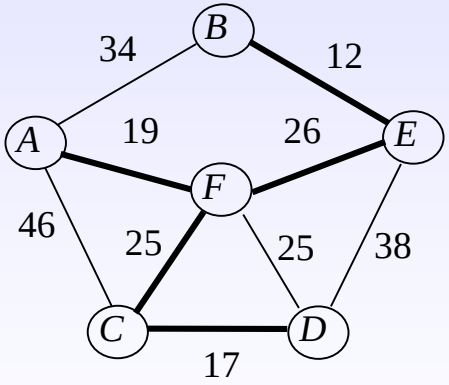
(d) $U=\{A, F, C, D\}$

cost={(A, B)34, **(F, E)26**}



(e) $U=\{A, F, C, D, E\}$

cost={**(E, B)12**}



(f) $U=\{A, F, C, D, E, B\}$

cost={ }

Prim算法构造最小生成树的过程示意

设图G中顶点的编号为0～n−1，Prim算法如下：

算法——Prim算法

1. 初始化两个辅助数组lowcost和adjvex；

2. U={u_0}; 输出顶点u_0；　//将顶点u_0加入生成树中

3. 重复执行下列操作n-1次

3.1 在lowcost中选取最短边，取adjvex中对应的顶点序号k；

3.2 输出顶点k和对应的权值；

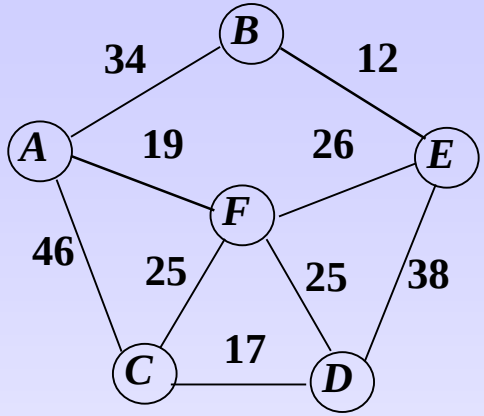
3.3 U=U+{k}；

3.4 调整数组lowcost和adjvex；

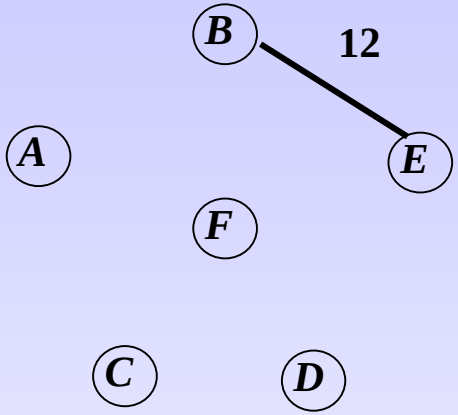
设连通网中有*n*个顶点，则第一个进行初始化的循环语句需要执行*n*−1次，第二个循环共执行*n*−1次，内嵌两个循环，其一是在长度为*n*的数组中求最小值，需要执行*n*−1次，其二是调整辅助数组，需要执行*n*−1次，所以，Prim算法的时间复杂度为$O(n^2)$。

（2）最短边策略：设$G=(V, E)$是一个无向连通网，令$T=(U, TE)$是G的最小生成树。最短边策略从$TE=\{\}$开始，每一次贪心选择都是在边集E中选取最短边(u, v)，如果边(u, v)加入集合TE中不产生回路，则将边(u, v)加入边集TE中，并将它在集合E中删去。

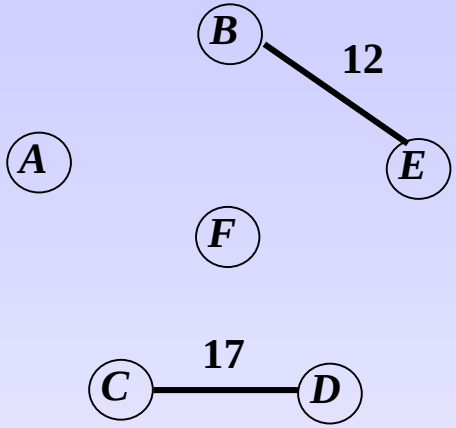
Kruskal算法就应用了这个贪心策略，它使生成树以一种随意的方式生长，先让森林中的树木随意生长，每生长一次就将两棵树合并，到最后合并成一棵树。



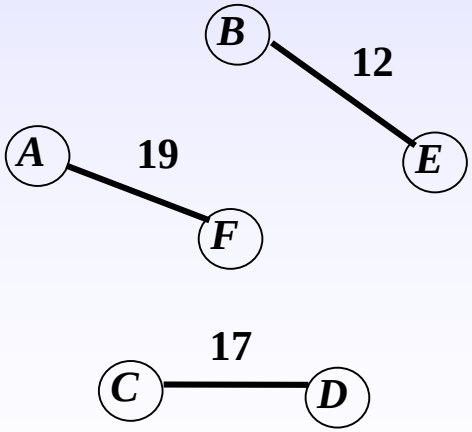
(a)



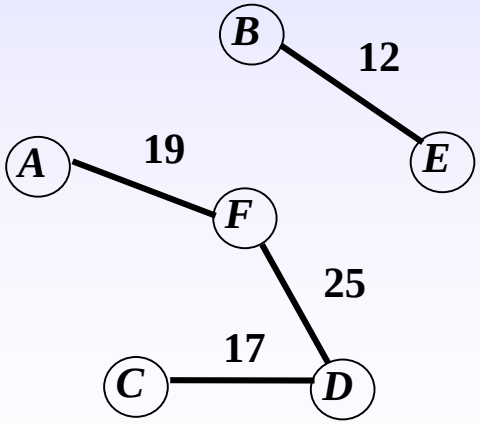
(b)



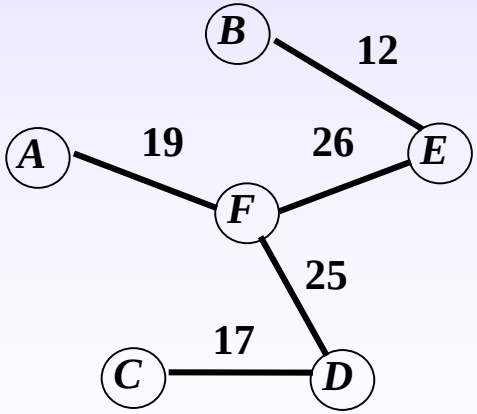
17 (c)



(d)



(e)



(f)

Kruskal方法构造最小生成树的过程

算法——Kruskal算法

1. 初始化：U=V；　TE={ }；
2. 循环直到T中的连通分量个数为1
　2.1 在E中寻找最短边(u，v)；
　2.2 如果顶点u、v位于T的两个不同连通分量，则
　　2.2.1 将边(u，v)并入TE；
　　2.2.2 将这两个连通分量合为一个；
　2.3　E=E-{(u，v)}；

Kruskal算法为了提高每次贪心选择时查找最短边的效率，可以先将图G中的边按代价从小到大排序，则这个操作的时间复杂度为$O(e\log_2 e)$，其中e为无向连通网中边的个数。对于两个顶点是否属于同一个连通分量，可以用并查集的操作将其时间性能提高到$O(n)$，所以，Kruskal算法的时间性能是$O(e\log_2 e)$。

3. 组合问题中的贪心法

背包问题

活动安排问题

多机调度问题

背包问题

给定n种物品和一个容量为C的背包，物品i的重量是w_i，其价值为v_i，背包问题是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?

设x_i表示物品i装入背包的情况，根据问题的要求，有如下约束条件和目标函数：

$$\begin{cases} \sum_{i=1}^{n} w_i x_i = C \\ 0 \le x_i \le 1 \quad (1 \le i \le n) \end{cases} \qquad \text{（式7.1）}$$

$$\max \sum_{i=1}^{n} v_i x_i \qquad \text{（式7.2）}$$

于是，背包问题归结为寻找一个满足约束条件式7.1，并使目标函数式7.2达到最大的解向量$X=(x_1, x_2, \ldots, x_n)$。

至少有三种看似合理的贪心策略：

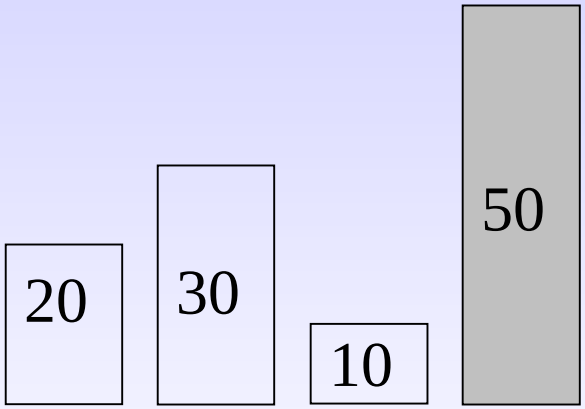
（1）选择价值最大的物品，因为这可以尽可能快地增加背包的总价值。但是，虽然每一步选择获得了背包价值的极大增长，但背包容量却可能消耗得太快，使得装入背包的物品个数减少，从而不能保证目标函数达到最大。

（2）选择重量最轻的物品，因为这可以装入尽可能多的物品，从而增加背包的总价值。但是，虽然每一步选择使背包的容量消耗得慢了，但背包的价值却没能保证迅速增长，从而不能保证目标函数达到最大。

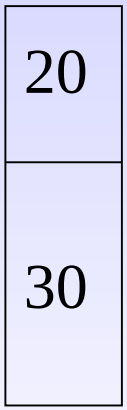
（3）选择单位重量价值最大的物品，在背包价值增长和背包容量消耗两者之间寻找平衡。

应用第三种贪心策略，每次从物品集合中选择单位重量价值最大的物品，如果其重量小于背包容量，就可以把它装入，并将背包容量减去该物品的重量，然后我们就面临了一个最优子问题——它同样是背包问题，只不过背包容量减少了，物品集合减少了。因此背包问题具有最优子结构性质。

例如，有3个物品，其重量分别是{20, 30, 10}，价值分别为{60, 120, 50}，背包的容量为50，应用三种贪心策略装入背包的物品和获得的价值如图所示。

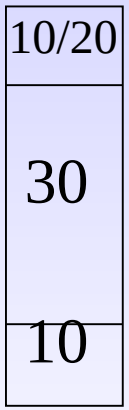


(a) 三个物品及背包



(b) 贪心策略1

(c) 贪心策略2



(d) 贪心策略3

设背包容量为C，共有n个物品，物品重量存放在数组w[n]中，价值存放在数组v[n]中，问题的解存放在数组x[n]中。

算法7.6——背包问题

```
1．改变数组w和v的排列顺序，使其按单位重量价值v[i]/w[i]降序排列；
2．将数组x[n]初始化为0；  //初始化解向量
3．i=1;
4．循环直到(w[i]>C)
     4.1 x[i]=1;   //将第i个物品放入背包
     4.2 C=C-w[i];
     4.3 i++;
5. x[i]=C/w[i];
```

算法7.6的时间主要消耗在将各种物品依其单位重量的价值从大到小排序。因此，其时间复杂性为*O*(*n*log2*n*)。

多机调度问题

设有n个独立的作业$\{1, 2, \ldots, n\}$，由m台相同的机器$\{M_1, M_2, \ldots, M_m\}$进行加工处理，作业$i$所需的处理时间为$t_i$（$1 \le i \le n$），每个作业均可在任何一台机器上加工处理，但不可间断、拆分。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。

贪心法求解多机调度问题的贪心策略是最长处理时间作业优先，即把处理时间最长的作业分配给最先空闲的机器，这样可以保证处理时间长的作业优先处理，从而在整体上获得尽可能短的处理时间。按照最长处理时间作业优先的贪心策略，当$m \geq n$时，只要将机器i的$[0, t_i)$时间区间分配给作业i即可；当$m < n$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。

例：设7个独立作业{1, 2, 3, 4, 5, 6, 7}由3台机器{M_1, M_2, M_3}加工处理，各作业所需的处理时间分别为{2, 14, 4, 16, 6, 5, 3}。贪心法产生的作业调度如下：

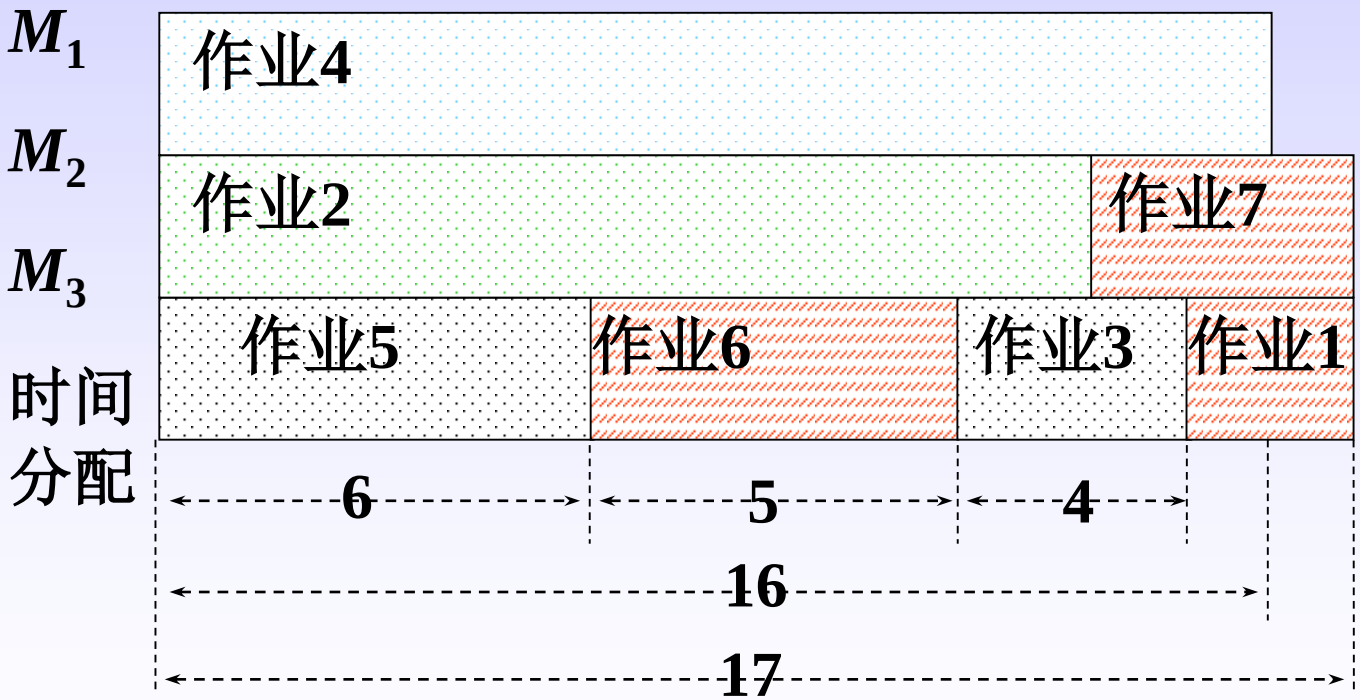


图7.10 三台机器的调度问题示例

算法——多机调度问题

```
1．将数组t[n]由大到小排序，对应的作业序号存储在数组p[n]中；
2．将数组d[m]初始化为0；
3．for (i=1; i<=m; i++)
    3.1 S[i]={p[i]};  //将m个作业分配给m个机器
    3.2 d[i]=t[i];
4. for (i=m+1; i<=n; i++)
    4.1  j=数组d[m]中最小值对应的下标；  //j为最先空闲的机器序号
    4.2  S[j]=S[j]+{p[i]};  //将作业i分配给最先空闲的机器j
    4.3  d[j]=d[j]+t[i];     //机器j将在d[j]后空闲
```

在算法7.9中，操作“数组d[m]中最小值对应的下标”如果采用蛮力法查找，则算法的时间性能为：

$$T(n)=\sum_{i=1}^{m}1+\sum_{i=m+1}^{n}m=m+(n-m)\times m$$

通常情况下$m<<n$，则算法7.9的时间复杂性为$O(n\times m)$。

练习

- 你的朋友要开一家安全公司，这需要得到n个不同的密码软件的许可证．根据规章，他们只能以每个月至多一个的速度得到这些许可证。每个许可证目前的售价是100美元，但他们全都按指数增长曲线变得更贵，设许可证j的费用每个月按照$r_j>1$的因子增加，且假设所有价格增长率是不同的，应该按照什么次序买这些许可证使花的总钱数最少？

思考题

- 让我们考虑一条长的乡村道路，沿着它非常稀疏地分布着一些房子。不管这种田园诗般的环境，假设所有这些房子的居民都是热心的蜂窝电话用户。你想把蜂窝电话基站放在沿着这条路的某些点上，使得每个房子都在一个基站的4英里之内。
- 给出一个有效算法，使用尽可能少的基站来实现这个目标。