算法设计与分析

贪心算法

主要内容

- 贪心算法的基本思想
- 活动安排问题
- 贪心策略的基本要素
- 贪心算法实例——单源最短路径
- 贪心算法实例——最小生成树
- 贪心算法实例——Huffman编码
- 贪心算法实例讨论
- 最小生成树的拓展

贪心算法的基本思想

- 适用于求解最优化问题的算法往往包含一系列步骤，每一步都有一组选择
- 贪心算法总是作出在当前看来是最好的选择
- 贪心算法并不从整体最优上加以考虑，它所作出的选择只是在某种意义上的局部最优选择

贪心算法的基本思想

- 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
- 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
- 与动态规划方法相比，贪心算法更简单，更直接

活动安排问题

- 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，如报告厅等
  - 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$s_j \geq$fi时，活动i与活动j相容
  - 目标：要在所给的活动集合中选出最大的相容活动子集合

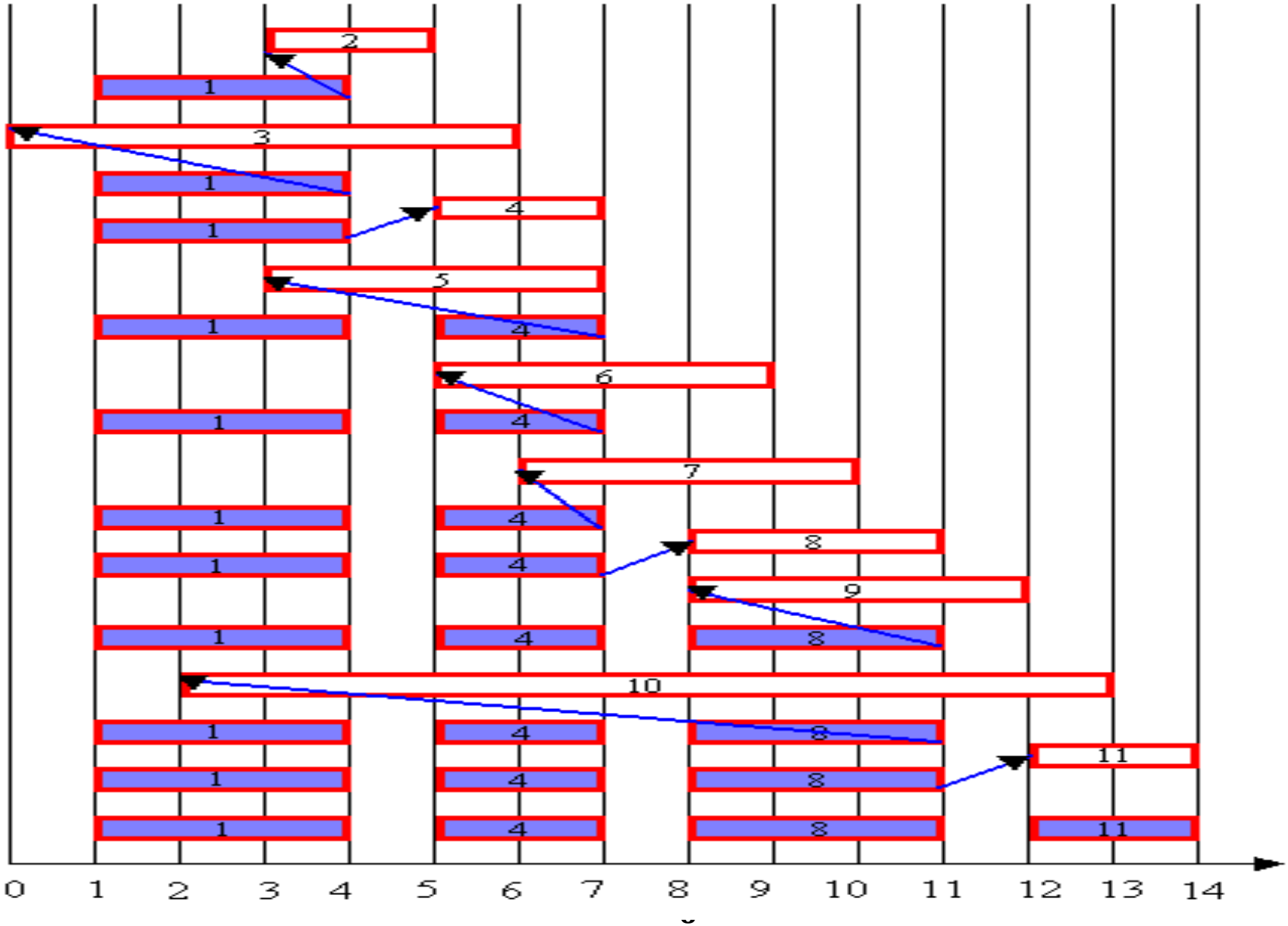
活动安排问题

- 贪心策略求解方案
  - 将活动按照结束时间的增序$f_1 \leq f_2 \leq \ldots \leq f_n$排列
  - 一开始选择活动1，然后依次检查后面的活动i是否与前面已选择的活动相容，若相容，则将活动i加入到已选择的活动集合A中；若不相容，则继续检查下一活动与已选择活动的相容性
  - 由于活动已按结束时间排序，设刚加入已选择活动集合的活动是j，因此只需检查活动i的是否与j相容；也就是检查是否满足$s_i \geq f_j$，若满足，则活动i与j相容
  - 贪心体现在总是选择具有最早完成时间的相容活动

活动安排问题

- 设待安排的11个活动的开始时间和结束时间已按结束时间的增序排列：

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


2
1
3
1
4
1
5
1
4
6
1
4
7
1
4
8
1
4
9
1
4
8
10
1
4
8
11
1
4
8
11
1
4
8
11
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

活动安排问题

GREEDY-ACTIVITY-SELECTOR(s, f)

```
1 n ← length[s]
2 A ← {activity 1}
3 j← 1
4 for i ← 2 to n
5     do if s_i ≥ f_j
6           then A ← A ∪ {activity i}
7                j← i
8 return A
```

活动安排问题

- 复杂度分析
  - 如果已经排序，算法的时间复杂度为Θ(n)
  - 如果事先没有按照结束时间增序排列，排序需O(nlgn)
- 贪心算法可以获得该问题的整体最优解
  - 可证明：1.活动安排问题有一个最优解以贪心选择开始；
  - 2.做出贪心选择之后，原问题简化为比原问题更小的但与原问题形式相同的子问题。

贪心算法的基本要素

- 贪心算法通过做一系列的选择来给出某一问题的最优解。它所作出的每一个选择当前状态下的最好选择（局部），即贪心选择
- 这种贪心选择并不总能产生最优解，但对于一些问题，比如活动安排问题，可以给出最优解

贪心算法的基本要素

- 可以根据下列步骤设计贪心算法
  - 将最优化问题转化为这样的一个问题，即先做出选择，再解决剩下的一个子问题
  - 证明原问题总有一个最优解是做贪心选择得到的，从而说明贪心选择的安全
  - 说明在做出贪心选择之后，子问题的最优解与所作出的贪心选择联合起来，可以得出原问题的一个最优解

贪心算法的基本要素

- 许多可以用贪心算法求解的问题，具备以下两个性质
  - 贪心选择性质
  - 最优子结构性质

贪心算法的基本要素

- 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
  - 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

贪心算法的基本要素

- 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

贪心算法的基本要素

- 0-1背包问题
  - 给定n种物品和一个背包。物品i的重量是Wi，其价值为Vi，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?
  - 限制：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

贪心算法的基本要素

- 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

贪心算法的基本要素

- 0-1背包问题和背包问题的求解分析
  - 这2类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

贪心算法的基本要素

- 用贪心算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Vi/Wi
  - 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

贪心算法的基本要素

- 对于0-1背包问题，贪心选择不能得到最优解
  - 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单位重量背包空间的价值降低
  - 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（子问题重叠）
  - 这正是该问题可用动态规划算法求解的重要特征

最优装载问题

- 问题
  - 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Wi。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
- 算法
  - 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

贪心算法的重要实例

- 几个重要的图算法
  - 单源最短路径
    - Dijkstra算法
  - 最小生成树
    - Prim算法
    - Kruskal算法

单源最短路径 Dijkstra算法

- 单源最短路径问题
  - 给定带权有向图G =(V,E)，其中每条边的权是非负实数
  - 给定V中的一个顶点，称为源
  - 要求计算从源到所有其他各顶点的最短路长度（这里路的长度是指路上各边权之和）

单源最短路径 Dijkstra算法

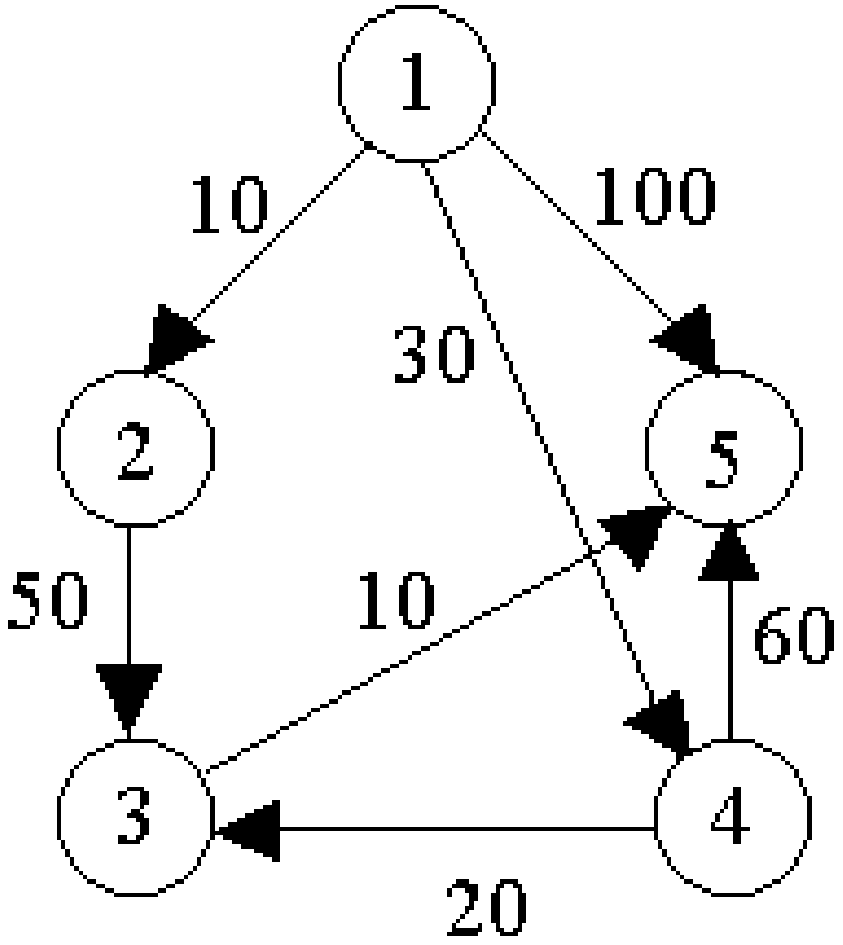
- 基本思想
  - 设置顶点集合S，初始时，S中仅含有源，此后不断作贪心选择来扩充这个集合
  - 一个顶点属于集合S当且仅当从源到该顶点的最短路径长度已知
  - 设u是G的某一个顶点，把从源到u且中间只经过S中顶点的路称为从源到u的特殊路径，并用数组dist记录当前每个顶点所对应的最短特殊路径长度
  - Dijkstra算法每次从V-S中取出具有最短特殊路长度的顶点u，将u添加到S中，同时对数组dist作必要的修改，检查dist(u)+[u,j]与dist[j]的大小，若dist(u)+[u,j]较小，则更新
  - 一旦S包含了所有V中顶点，dist就记录了从源到所有其他顶点之间的最短路径长度

单源最短路径 Dijkstra算法

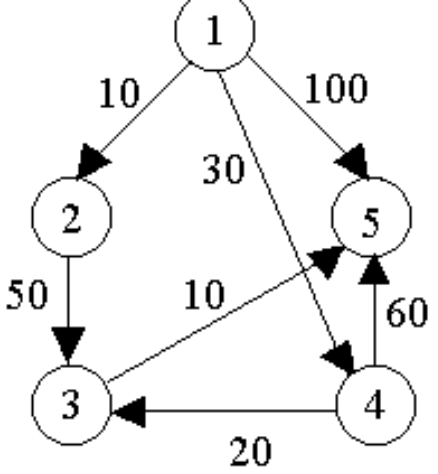
- 贪心策略
  - 因为Dijkstra算法总是在V-S中选择“最近”（具有最短特殊路长度）的顶点，所以说Dijkstra算法使用了贪心策略

单源最短路径 Dijkstra算法

- 实例，源点1



单源最短路径 Dijkstra算



●实例：Dijkstra算法的计算过程：

迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

单源最短路径 Dijkstra算法

- 分析
- 贪心策略体现在对V-S中的点的选择上
- 时间复杂度
  - 对于具有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，$O(n^2)$

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

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
- 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
  - 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
- 用贪心算法设计策略可以设计出构造最小生成树的有效算法
- 构造最小生成树的Prim算法和Kruskal算法用到了以上性质

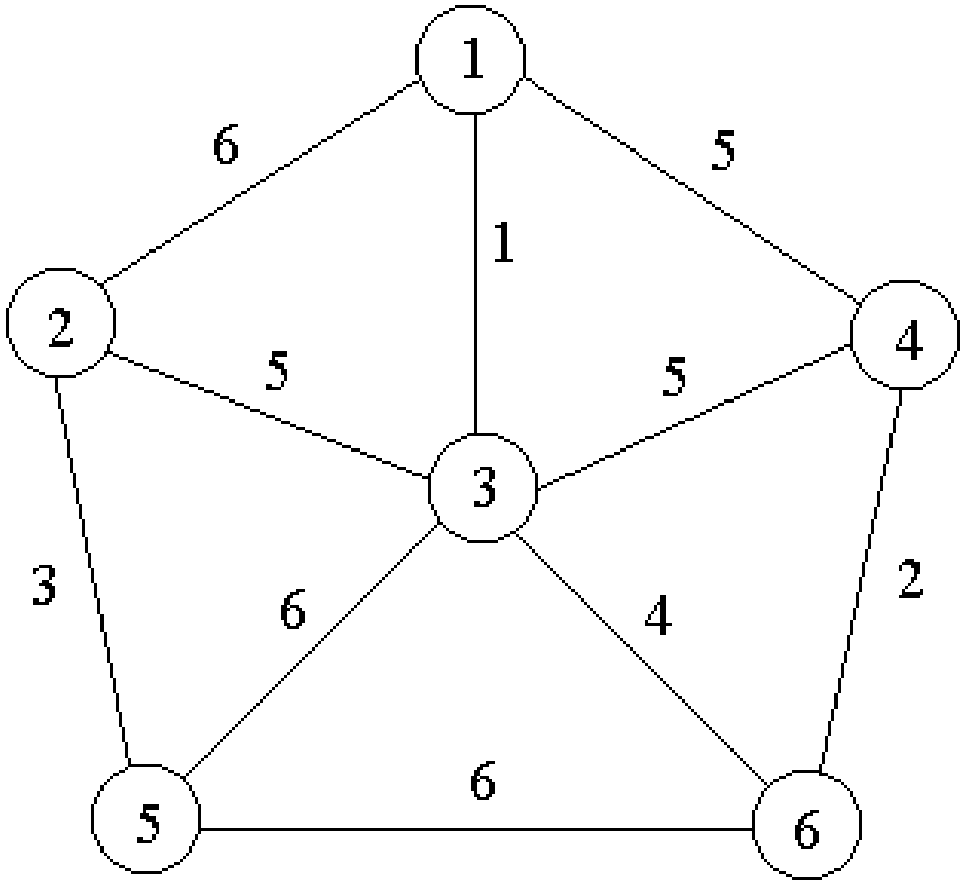
最小生成树 Prim算法

- 设G=(V,E)是连通带权图，V={1,2,…,n}
- Prim算法的基本思想
  - 首先置S={1}
  - 然后，只要S是V的真子集，就作如下的贪心选择：

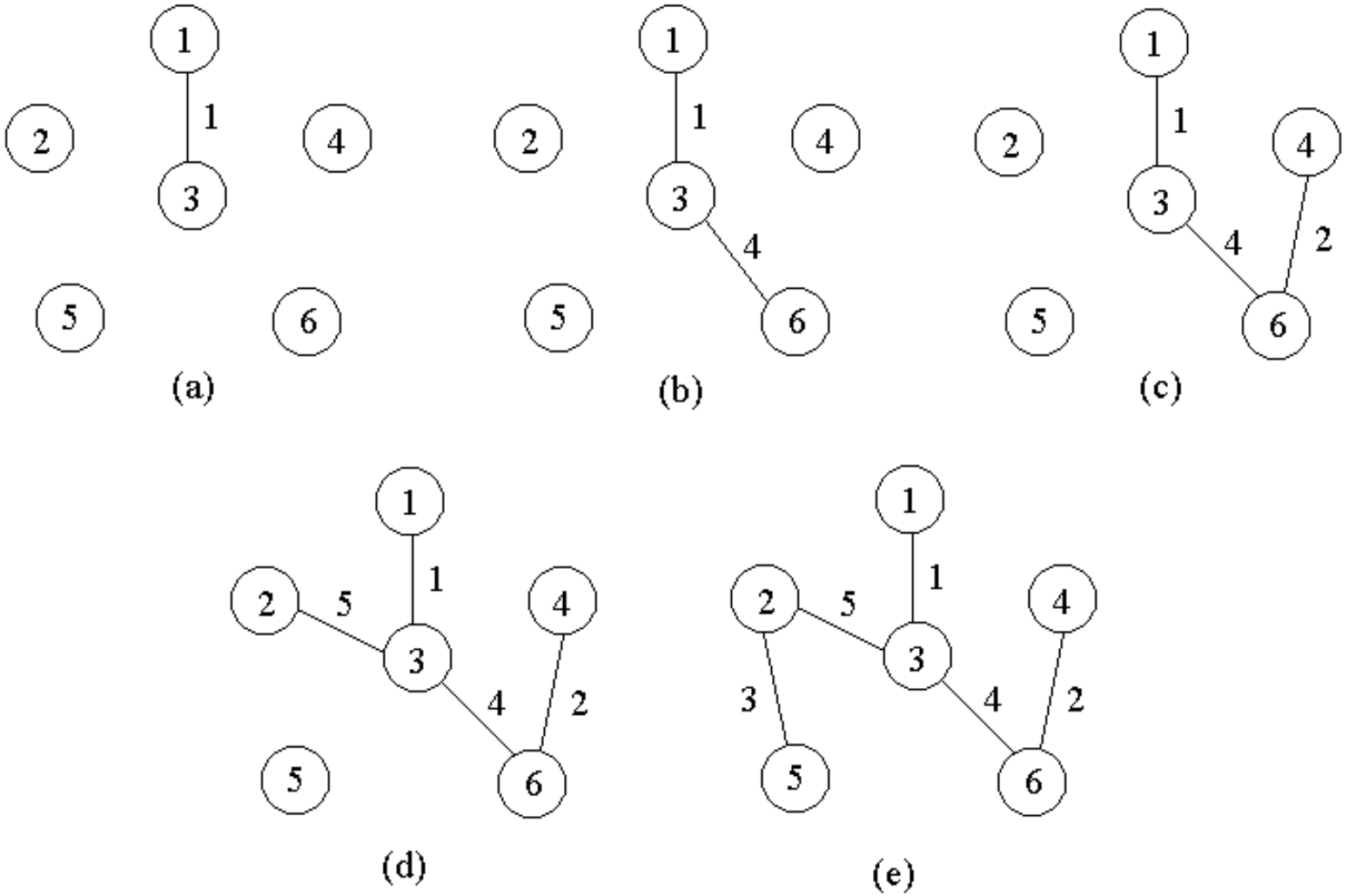
    选取满足条件i∈S，j∈V-S，且c[i][j]最小的边，将顶点j添加到S中
  - 这个过程一直进行到S=V时为止
  - 在这个过程中选取到的所有边恰好构成G的一棵最小生成树

最小生成树 Prim算法

- Prim算法求解实例



最小生成树 Prim算法



最小生成树 Prim算法

- 在上述Prim算法中，还应当考虑如何有效地找出满足条件$i \in S, j \in V-S$，且权c[i][j]最小的边(i,j)，实现这个目的的较简单的办法是设置2个数组closest和lowcost
- closest[j]是j在S中的邻接顶点，它与j在S中的其它邻接顶点k相比有c[j][closest[j]] ≤c[j][k]
- lowcost[j]的值就是c[j][closest[j]]

最小生成树

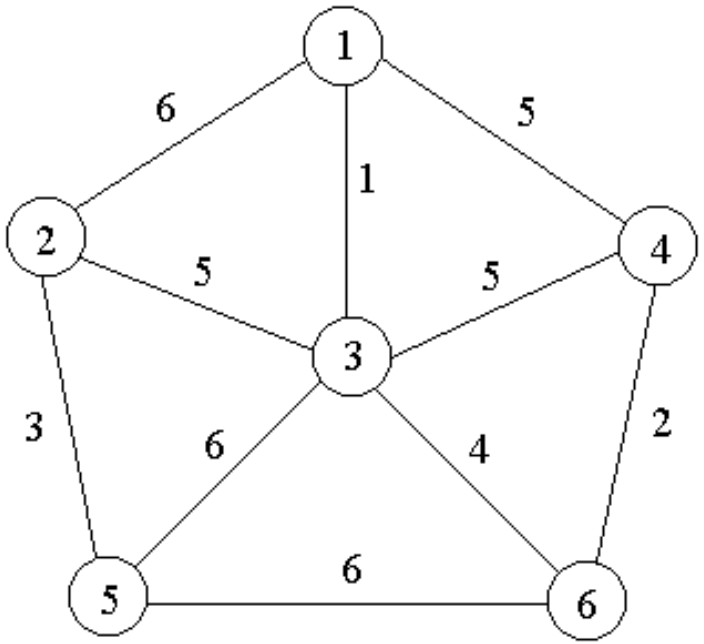
- 在Prim算法执行过程中，先找出V-S中使lowcost值最小的顶点j，然后根据数组closest选取边(j,closest[j])，最后将j添加到S中，并对closest和lowcost作必要的修改
- 用这个办法实现的Prim算法所需的时间复杂度为$O(n^2)$

最小生成树 Kruskal算法

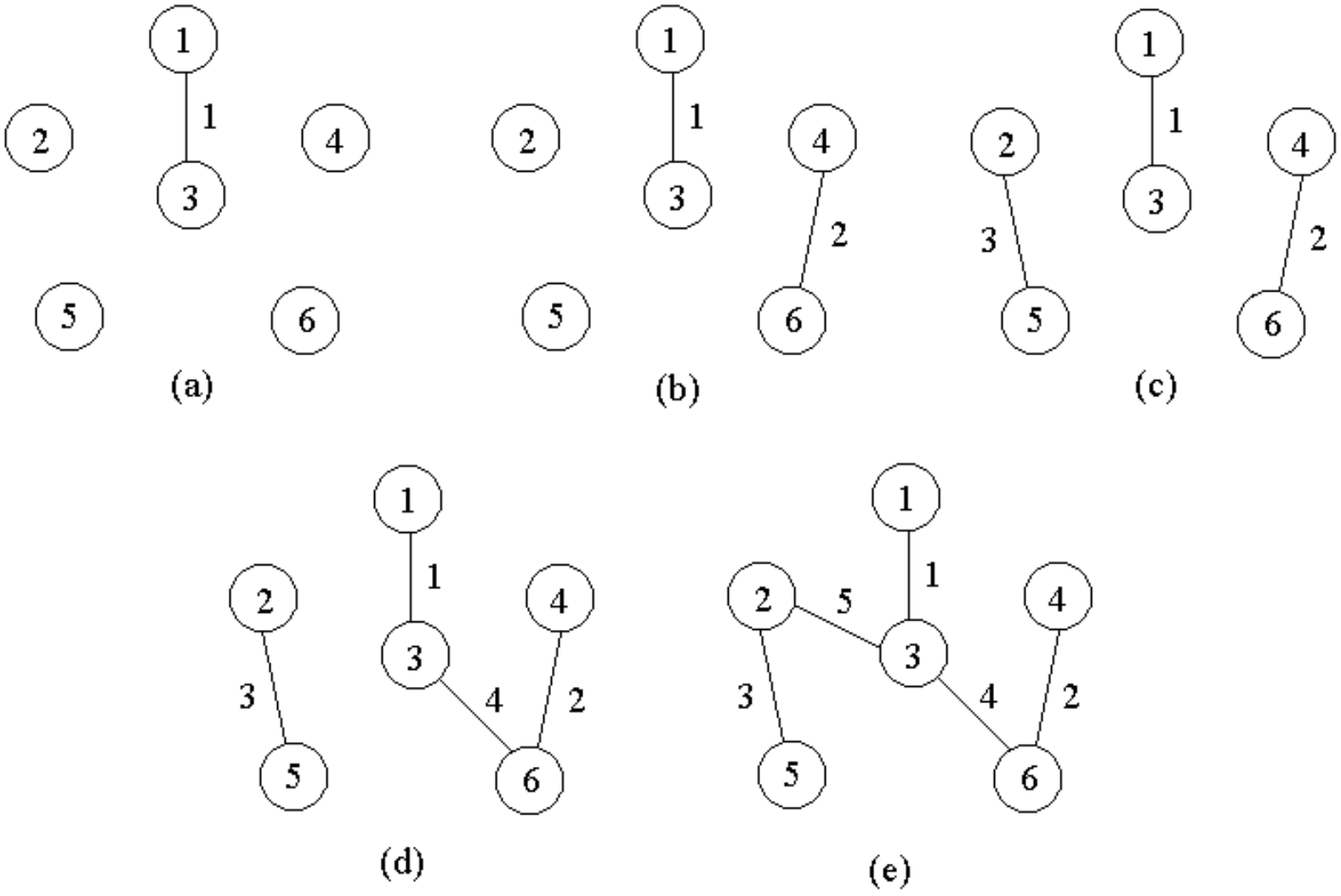
- Kruskal算法的基本思想是
  - 首先将G的n个顶点看成n个孤立的连通分支。将所有的边按权值从小到大排序
  - 然后从第一条边开始，依边权递增的顺序查看每一条边，并按下述方法连接2个不同的连通分支：当查看到第k条边(v,w)时，如果端点v和w分别是当前2个不同的连通分支T1和T2中的顶点时，就用边(v,w)将T1和T2连接成一个连通分支，然后继续查看第k+1条边；如果端点v和w在当前的同一个连通分支中，就直接再查看第k+1条边
  - 这个过程一直进行到只剩下一个连通分支时为止

最小生成树

- Kruskal算法的贪心策略
  - 将图中的边按权值由小到大排序，由小到大顺序选取各条边，若选某边后不形成回路，则将其保留作为树的一条边



最小生成树



Huffman编码

- Huffman编码
  - 一种可变字长编码(VLC)方式
  - Huffman于1952年提出
  - 该方法依据字符出现的概率来构造字符编码，一般称为Huffman编码
  - 广泛应用于数据文件的压缩

Huffman编码

- 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他字符代码的前缀
  - 编码的前缀性质可以使译码方法非常简单
  - 表示最优前缀码的二叉树中任一结点都有2个儿子结点
  - 平均码长定义为 $B(T)=\sum_{c\in C} f(c)d_T(c)$
  - 使平均码长达到最小的前缀码编码方案称为给定编码字符集C的最优前缀码

Huffman编码

- 构造方式
  - Huffman提出构造最优前缀码的贪心算法，由此产生的编码方案称为Huffman编码
  - Huffman算法以自底向上的方式构造表示最优前缀码的二叉树T
  - 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

Huffman编码

- 构造过程
  - 编码字符集C中每一字符c的频率是f(c)
  - （1）根据给定的|C|个频率，构造n棵二叉树的集合$F=\{T_1,T_2,...,T_n\}$，其中T_i中只有一个根结点，左右子树为空；
  - （2）在F中选取两棵根结点频率最小的树作为左、右子树构造一棵新的二叉树，且置新的二叉树的根结点的频率为左、右子树上根结点的频率之和；
  - （3）将新的二叉树加入到F中，删除原两棵根结点频率最小的树；
  - （4）重复（2）和（3）直到F中只含一棵树为止

Huffman编码

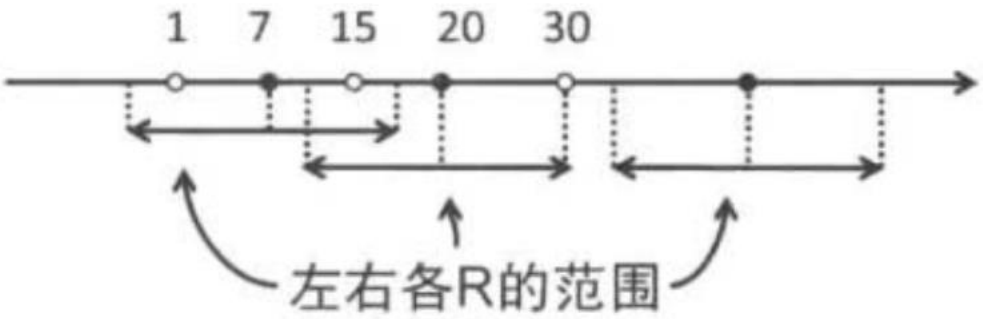
- 贪心策略
  - 每次都选择根节点“频率”值最小的两棵树合并

Saruman's Army

- 直线上有N个点。点i的位置是X_i。从这N个点中选择若干个，给它们加上标记。
- 对每一个点，其距离为R以内的区域里必须有带有标记的点(自己本身带有标记的点，可以认为与其距离为0的地方有一个带有标记的点)。
- 在满足这个条件的情况下，希望能为尽可能少的点添加标记。
- 请问至少要有多少点被加上标记?

Saruman's Army

- ○ 普通的点
- ● 带有标记的点



添加标记的例子

输入

```
N = 6
R = 10
X = {1, 7, 15, 20, 30, 50}
```

输出

3（如上图所示）

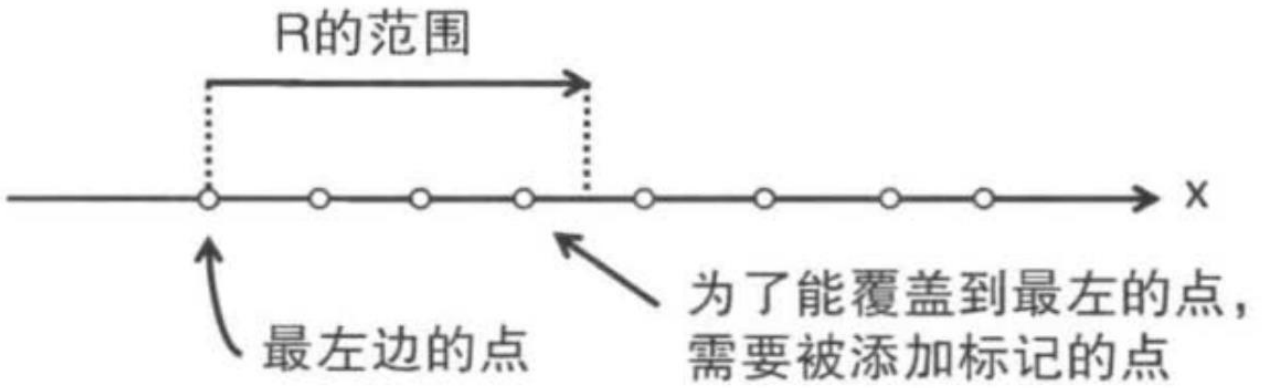
⚠限制条件

- $1 \leqslant N \leqslant 1000$
- $0 \leqslant R \leqslant 1000$
- $0 \leqslant X_i \leqslant 1000$

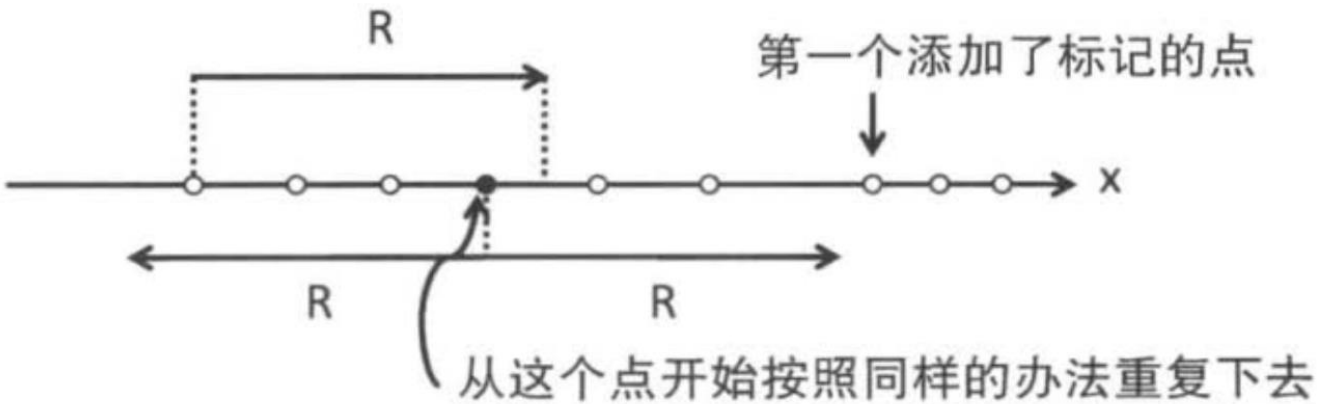
Saruman's Army

- 目标：用最少的点做标记。
- 思路：
  - 从最左边的点开始考虑。对于这个点，到距其R以内的区域内必须要有带有标记的点。(此点位于最左边,所以显然)带有标记的这个点一定在此点右侧(包含这个点自身)。
  - 究竟要给哪个点加上标记呢?答案应该是从最左边的点开始，距离为R以内的最远的点。因为更左的区域没有覆盖的意义，所以应该尽可能覆盖更靠右的点。
  - 如上所示，加上了第一个标记后，剩下的部分也用同样的办法处理。对于添加了符号的点右侧相距超过R的下一个点，采用同样的方法找到其右侧R距离以内最远的点添加标记。在所有的点都被覆盖之前不断地重复这一过程。

Saruman's Army



对于最左边的点的观察



反复操作的情况

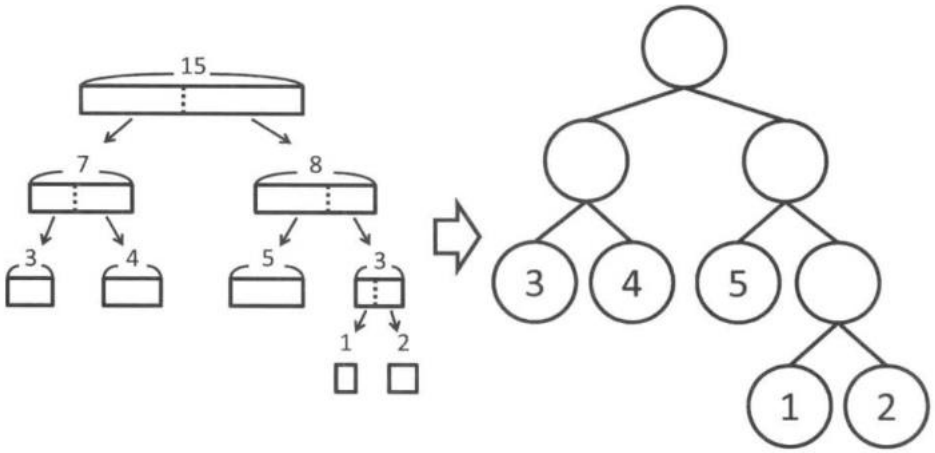
Fence Repair

- 农夫约翰为了修理栅栏，要将一块很长的木板切割成N块。准备切成的木板的长度为L_1、L_2、…、L_N，未切割前木板的长度恰好为切割后木板长度的总和。
- 每次切断木板时，需要的开销为这块木板的长度。
- 例如长度为21的木板要切成长度为5、8、8的三块木板。长21的木板切成长为13和8的板时，开销为21。再将长度为13的板切成长度为5和8的板时，开销是13。于是合计开销是34.
- 请求出按照目标要求将木板切割完最小的开销是多少。
- 例：
  - 输入：N=3, L={8, 5, 8}
  - 输出：34【具体见上述描述】

Fence Repair

- 由于木板的切割顺序不确定，自由度很高，看似很难入手。
- 可以用贪心策略？

Fence Repair



对应切割方法的二叉树的例子

- 每一个叶子节点就对应了切割出的一块块木板。叶子节点的深度就对应了为了得到对应木板所需的切割次数，开销的合计就是各叶子节点的木板的长度×节点的深度的总和。
- 上图示例的全部开销就可以这样计算
  - 3×2+4×2+5×2+1×3+2×3=33

Fence Repair

- 最佳切割方法首先应该具有如下性质
  - 最短的板与次短的板的节点应当是兄弟节点
- 对于最优解来讲，最短的板应当是深度最大的叶子节点之一。所以与这个叶子节点同一深度的兄弟节点一定存在，并且由于同样是最深的叶子节点，所以应该对应于次短的板。

Fence Repair

- 将L按照大小顺序排列，那么最短的板应该是L_1而次短的则是L_2。如果它们在二叉树中是兄弟节点，就意味着它们是从一块长度为(L_1+L_2)的板切割得来的。由于切割顺序是自由的，不妨当作是最后被切割。
- 这样一来，在这次切割前就有$(L_1+L_2),L_3,L_4,\ldots,L_N$这样的$N$-1块木板存在。递归地将这$N$-1块木板的问题求解，就可以求出整个问题的答案。

田忌赛马

- 田忌和齐王赛马，现给出双方马的速度，赢一局得200铜钱
- 问：田忌最多会赢得多少铜钱？
- 如何求解？
  - 二分图匹配算法？
  - 动态规划方法？
  - 贪心策略？

田忌赛马

- 贪心策略的思路
  - 首先，将田忌和齐王的马按马的速度递增顺序分别排列，得到递增序列A和B，其田忌的马为$A=a_1\cdot\cdot a_n$，齐王的马为$B=b1..b_n$。
  - 若田忌最慢的马快于齐王最慢的马($a_1>b_1$)，则将a_1和b_1比，因为齐王最慢的马一定输，输给田忌最慢的马a_1合适。
  - 若田忌最慢的马慢于齐王最慢的马($a_1<b_1$)，则将a_1和b_n比，因为a_1一定会输，输给齐王最快的马合适。
  - 若田忌最快的马快于齐王最快的马($a_n>b_n$)，则将a_n和b_n比，因为a_n一定赢，赢齐王最快的马合适。
  - 若田忌最快的马慢于齐王最快的马($a_n<b_n$)，则将a_1和b_n比，因为b_n一定赢，赢田忌最慢的马合适。
  - 田忌最慢的马和齐王最慢的马的速度相等($a_1=b_1$)，并且田忌最快的马比齐王最快的马快(a.>b.)时，将a,和b,比。
  - 田忌最快的马和齐王最快的马的速度相等($a_n=b_n$)时，则a_1将和b_n比有最优解。
  - 上述贪心策略给出了田忌赛马的过程。

最小生成树的拓展

- 最小瓶颈生成树
- 次小生成树
- 增量最小生成树
- 最小k度限制生成树
- 最优比率生成树

最小瓶颈生成树

- 给出加权无向图，求一棵生成树，使得最大边权值尽量小。
- 边的最大值最小的生成树

最小瓶颈生成树

- 目标关注的是最大边的权值
- 求解思路：
  - 可以从一个空图开始，按照权值从小到大的顺序依次加入各条边，则图第一次连通时，该图的最小生成树就是原图的最小瓶颈生成树。
  - Kruskal算法？原图的最小生成树就是一棵最小瓶颈生成树
  - 请注意：并非每棵最小瓶颈生成树都是最小生成树

次小生成树

- 除了最小生成树之外的具有最小总权重的生成树
- 注意，如果最小生成树不唯一，次小生成树的权值与最小生成树相同

次小生成树

- 次小生成树不会与最小生成树完全相同，因此可以枚举最小生成树中不在次小生成树中出现的边。
- 注意最小生成树只有n-1条边，所以只需枚举n-1次。
- 每次在剩下的边里求一次最小生成树，则这n-1棵“缺一条边的图”的最小生成树中权最小的就是原图的次小生成树。

增量最小生成树

- 从包含n个点的空图开始，依次加入m条带权边。
- 每加入一条边，输出当前图中的最小生成树权值（如果当前图不连通，则输出无解）。

增量最小生成树

- 每次加入新边后，重新求完整的最小生成树问题？
- 可以完成，时间复杂度高
- 改进：
  - 每次求出新的最小生成树后，把其它的边删除
  - 这样每次只需计算一个n条边（原生成树有 n - 1条，新加入一条）的图的最小生成树

增量最小生成树

- 进一步改进
  - 加入一条边$e = (u, v)$之后，图中恰好包含一个环。
  - 删除该回路上权值最大的边即可，因此只需在加边之前的MST上找到u到v的唯一路径上的权值最大的边，再和e比较，删除权值较大的一条。

最小k度限制生成树

- 实践中的最小生成树问题，由于问题背景和适用环境的要求，有时需要加入一些限制条件。
- 比如，要求生成树中某一节点的度为k。
- 最小k度限制生成树，就是指定生成树中某一点的度为k的最小生成树。
- 可以思考如何求解

最优比率生成树

- 在现实中，还会碰到要使得花费和收益的比最小化的问题。
- 最优比率生成树： 带权无向图$G(V, E)$，对于G中的每条边，存在两个参数benefit[i]（收入）和cost[i]（花费），分别表示为b_i和c_i，要求计算一颗生成树T，使得$\Sigma x_i b_i / \Sigma x_i c_i$最大或使得$\Sigma x_i c_i / \Sigma x_i b_i$最小，即达到最大收益；$x_i$表示第i条边是否包含在$T$中，是则为1，否则为0。这样的生成树称为最优比率生成树。
- 可以思考如何求解