Ch4 贪心算法

1. 基本概念

定义

一个算法通过一些小的步骤来建立一个解，每一步根据局部情况选择一个决定使得某些主要的指标能够得到优化

同一个问题往往可以涉及不同的贪心算法，每个算法局部的、增量的优化求解过程中的某个不同的量

证明贪心是最优解

贪心算法领先 Greedy algorithm stays ahead

- 如果一个人以一步接一步的方法测量贪心算法的进展时，他会看到每一步做的都比其他的算法好，从而证明产生了最优解
- 证明了贪婪算法在每一步之后，所得的解不坏与其它方法所得到的解

交换论证 Exchange argument

- 对于这个问题考虑任何可能的解，逐渐把它转换成由贪心算法找到的解且不损害它的质量
- 在不影响其质量的前提下，将任何解逐步转化为由贪心算法找到的解

结构化 Structural

- 分析解的边界并且证明贪婪算法总是能达到这个边界，从而是最优的

2. 兑换硬币 Coin Changing

2.1 问题描述

给定硬币面值为：1，5，10，25，100，设计一种方法，使用最少的硬币支付指定的金额

2.2 算法

在每次迭代中，添加不超过支付金额的最大价值硬币

CASHIERS-ALGORITHM(x,c1,...,cn)

```
SORT n coin denominations so that c1 < c2 < ... < cn
S ← 0 // 准备支付的硬币的集合
while (x > 0){
    k ← largest coin denomination ck such that ck ≤ x
    if (no such k){
        return "no solution"
    }else{
        x ← x - ck
        S ← S ∪ {k}
    }
}
return S
```

该算法对于本题目是最优解，但是对于不一样面值的情况下不一定是最优解

2.3 证明-归纳法

贪心算法对于美国硬币1，5，10，25，100 是最优的

- 当要支付的金额介于$c_k \le x < c_{k+1}$时，贪心算法会取硬币 k
- 证明任何最优解中都会包含硬币 k
    - 如果不是，需要用更小的硬币 $c_1, \ldots, c_{k-1}$ 来支付总共 x 的金额
    - 下面的表显示没有可以完成这样的最优解

k	c_k	All optimal solutions must satisfy	Max value of coins 1, 2, ..., k-1 in any OPT
1	1	P ≤ 4	-
2	5	N ≤ 1	4
3	10	N + D ≤ 2	4 + 5 = 9
4	25	Q ≤ 3	20 + 4 = 24
5	100	no limit	75 + 24 = 99

- 问题减小到支付金额为 $x - c_k$，通过归纳法，问题被优化到一个更小的贪心算法中

2.4 时间复杂度分析

时间复杂度 $O(n\ logn) + |S|$ 其中 S 是贪心取的硬币数

注意：使用整数除法可以稍微加快速度

2.5 反例

贪婪算法对于美国邮政面值并不是最优解

1, 10, 21, 34, 70, 100, 350, 1225, 1500

140¢.

- 贪婪算法：100, 34, 1, 1, 1, 1, 1, 1
- 最优解：70, 70

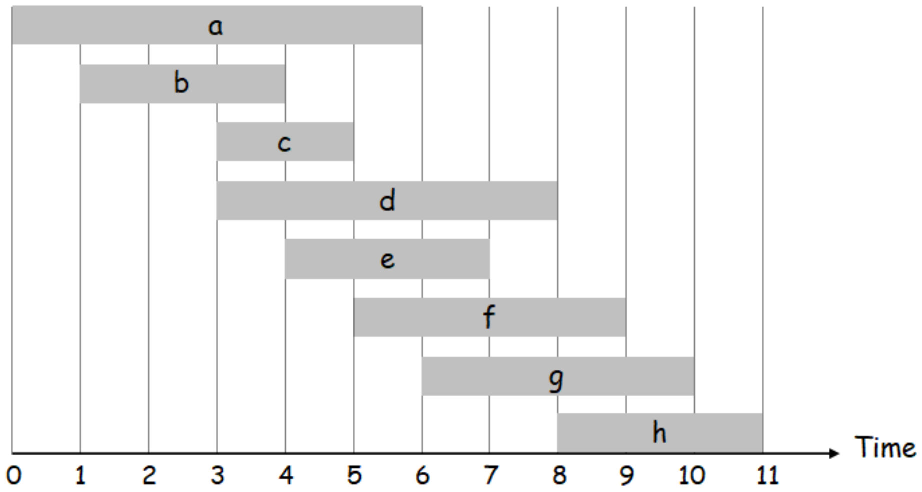
3. 区间调度 Interval Scheduling

3.1 问题描述

任务 j 的开始时间为 s_j，结束时间为 f_j

如果两个任务的时间没有重叠，我们说它们是相容 compatible 的

找到最大数量的相容任务的集合



最优解：B，E，H

3.2 算法

按照某种顺序对任务进行排序，然后依次安排相容的任务进行加工

- [最早开始时间优先算法]：按照开始时间 s_j 升序排列
- [最早结束时间优先算法]：按照结束时间 f_j 升序排列
- [最小区间优先算法]：按照时间长度升序排列 $f_j - s_j$
- [最小冲突优先算法]：计算每个任务与其它任务冲突的次数 c_j

我们举出特例后可以发现，最早结束时间优先算法可能是最优解

EARLEST-FINISH-TIME-FIRST(n,s1,...,sn,f1,...,fn)

```
// O(n logn)
SORT jobs by finish time so that f1 ≤ f2 ≤ ... ≤ fn
A = 0;
// O(n)
for (j = 1 to n) {
    if (job j is compatible with A) {
        A = A ∪ {j};
    }
}
return A
```

3.3 时间复杂度分析

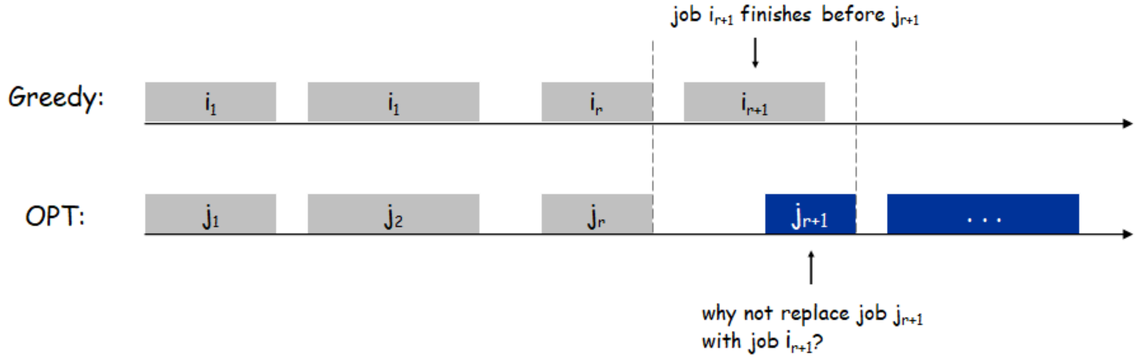
时间复杂度 $O(n\ logn)$

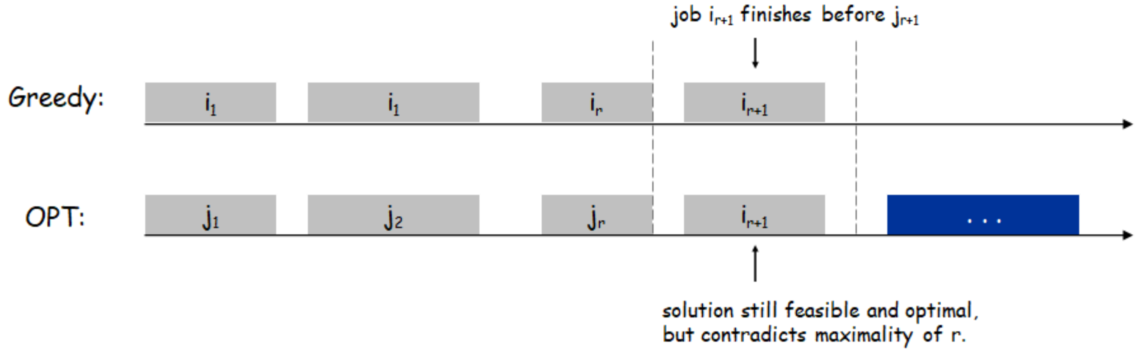
- 对所有任务进行排序 $O(n\ logn)$
- 比较加入的任务 $O(n)$

3.4 证明-反证法

贪婪算法是最优解

- 假设贪婪算法不是最优的
- 让 $i_1, i_2, ..., i_k$ 表示通过贪婪算法选出来的任务的集合
- 令 $j_1, j_2, .., j_m$ 是最优解选出的集合，其中 $i_1 = j_1, i_2 = j_2, ..., i_r = j_r$ 对于某一个数 r
- 那么对于 j_{r+1}，它的结束时间要晚于 i_{r+1} 的结束时间，如果我们在最优解中，把 j_{r+1} 替换成 i_{r+1}，我们同样得到了一个相同目标值的可行解，并且它的开始时间早于 j_{r+1}





这个解与贪婪算法的前 $r+1$ 个解相同，与我们 r 的设定产生矛盾

4. 区间划分 Interval Partitioning

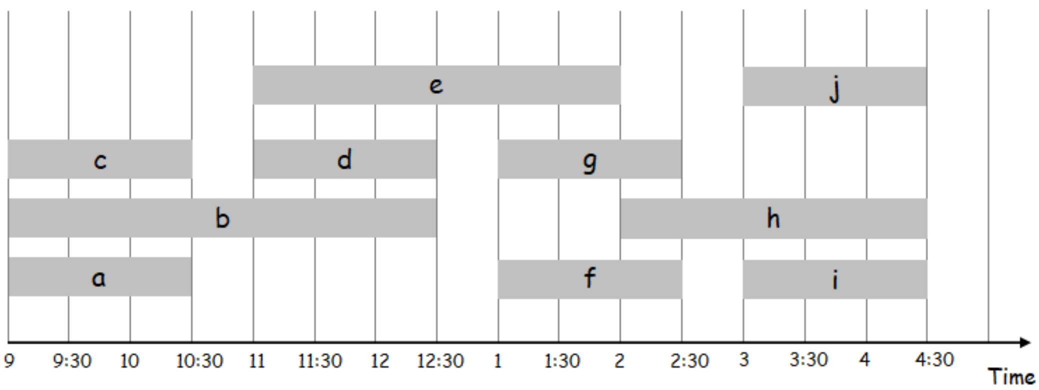
4.1 问题描述

一些讲座 j 有开始时间 s_j 和结束时间 f_j

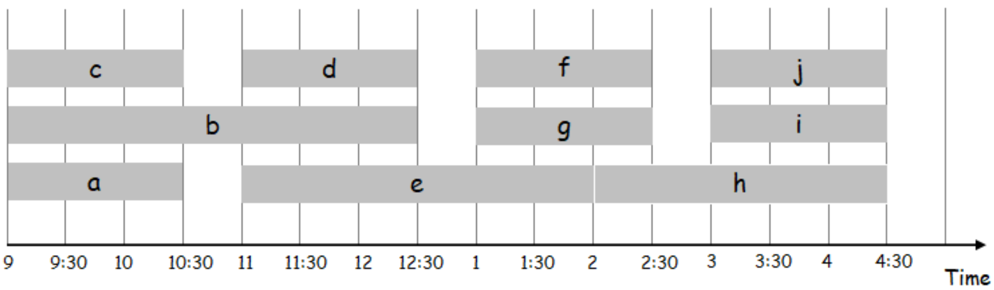
我们希望用最少的教室来安排这些讲座，并且不会有两个讲座同时出现在一个教室

示例：划分a-j 10个课程使用的教室

4个教室的情况



3个教室的情况

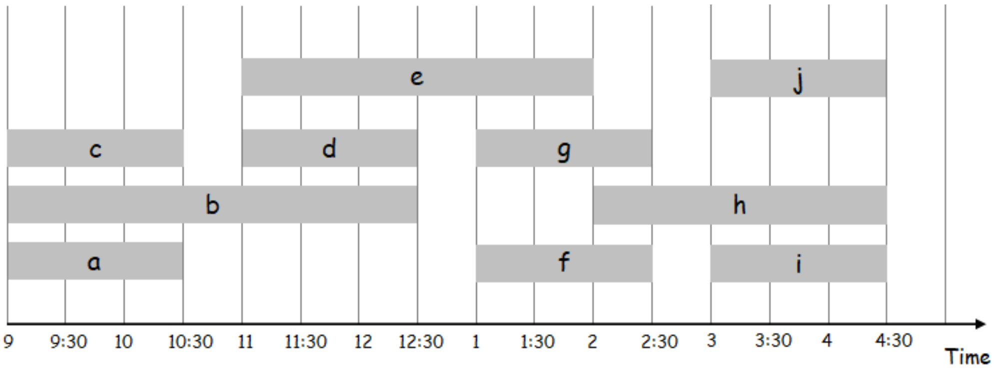


4.2 概念定义

区间深度 Depth

- 所有时刻中，包含某个时刻区间最多的数量

观察：需要教室的数量 ≥ 深度



- 如上图，区间深度为 3

4.3 算法

按照讲座的开始时间 s_j 升序排列

对于每个讲座，如果没有相容的教室，就开辟一个新的教室

EARLIST-START-TIME-FIRST(n,s1,...,sn,f1,...,fn)

```
// O(n logn)
SORT lectures by start time so that s1 ≤ s2 ≤ ... ≤ sn
d = 0;  //number of allocated classrooms
for (j = 1 to n){
    if (lecture j is compatible with some classroom) {
        Schedule lecture j in any such classroom k;
    }else{
        Allocate a new classroom d + 1;
        Schedule lecture j in classroom d + 1;
        d += 1;
    }
}
return schedule
```

4.4 时间复杂度分析

时间复杂度 $O(n\ logn)$

4.5 证明

贪心算法是最优解

- 让 d = 贪婪算法分配的班级数
- 为什么开放教室 d，是因为我们需要安排一个讲座，比如 j，它与所有 $d-1$ 其他教室不兼容
- 由于我们是按开始时间排序的，所以所有这些不兼容都是由不晚于 s_j 开始的讲课造成的
- 因此，我们有 d 个讲座在时间 $s_j+\epsilon$ 重叠
- 可以观察到，所有调度的使用 ≥ d 个教室

5. 最少延迟调度 Scheduling to Minimize Lateness

5.1 问题描述

一台生产设备每一个时刻可以处理一个任务

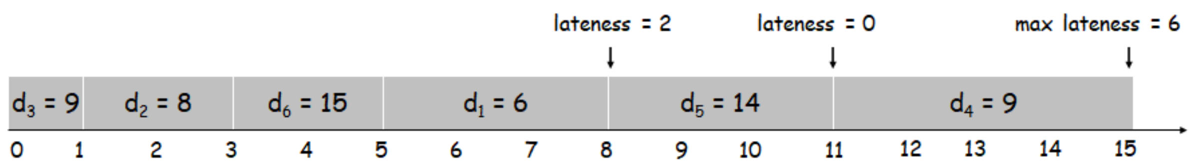
任务 j 需要 t_j 的加工时间，截止时间为 d_j （至少在 d_j 前完成）

如果任务 j 的开始时间为 s_j，它完成的时间 $f_j=s_j+t_j$

定义这个任务的延迟时间为 $l_j=max\{0,f_j-d_j\}$

对任务进行排序，使得任务的最大延迟时间 $L=max(l_j)$ 最小

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



5.2 算法

按一定的顺序排序任务

- [最短处理时间优先]：以处理时间 t_j 升序考虑工作

	1	2
t_j	1	10
d_j	100	10

- [最早截止时间优先]：按截止时间 d_j 的升序来考虑工作
- [最短宽恕时间]：考虑按 $d_j - t_j$ 升序排列的工作

	1	2
t_j	1	10
d_j	2	10

按照最早截止时间优先的算法，得到的结果是最好的

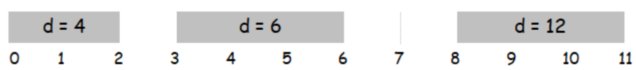
EARLEST-DADLINE-FIRST(n,t1,...,tn,d1,...,dn)

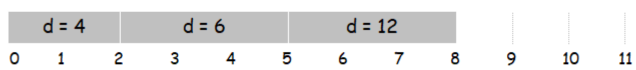
```
SORT jobs so that d1 ≤ d2 ≤ ... ≤ dn
t ← 0
for (j = 1 to n){
    Assign job j to interval [t, t + tj]
    sj ← t, fj ← t + tj
    t ← t + tj
}
return Intervals [s1,f1],[s2,f2],...,[sn,fn]
```

5.3 定义

最优排序没有空闲时间

一定存在一个无空闲时间的最优调度

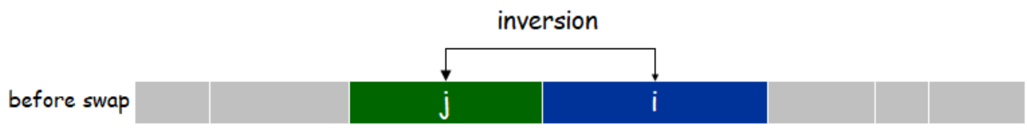




贪心算法没有空闲时间

逆序

在一个调度 S 中的一个逆序是指一对任务 i, j 使得 i 比 j 的截止时间要早（$d_i < d_j$），但是 j 比 i 调度时间靠前



贪婪算法不存在逆序

如果两个相邻的任务构成逆序，交换两个逆序任务，不会增加最大延迟

交换后

- 任务 i 的延迟 $l'_i \leq l_i$ 肯定不会增加
- 任务 j 的延迟 $l'_j = t_i - d_j \leq l_i = t_i - d_i$

5.4 证明-反证法

贪婪算法是最优解

- 令 $S*$ 是最优排序，并且包含最少的逆序
- 假设 $S*$ 没有空闲时间
- 如果 $S*$ 没有逆序，那么贪婪算法返回的解 $S = S*$
- 如果 $S*$ 有逆序，令 $i - j$ 是一个逆序
    - 交换 $i - j$ 不会增加最大延迟，但是会严格下降逆序数目
    - 这与 $S*$ 的设定产生矛盾

6. 最优缓存 Optimal Caching

6.1 问题描述

缓存中可以存储 k 个元素

有 m 个访问元素依次到达，记为 $d_1, d_2, \ldots, d_m$

如果缓存中存放了该元素，记为缓存命中

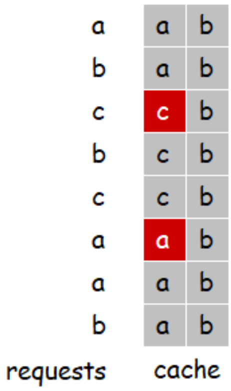
如果缓存中没有存放该元素，我们必须剔除某个已经存在的元素，加入新的要访问的元素

制定一个更新缓存的计划，使得剔除的操作次数最少

示例：k = 2，初始缓存 = a，b

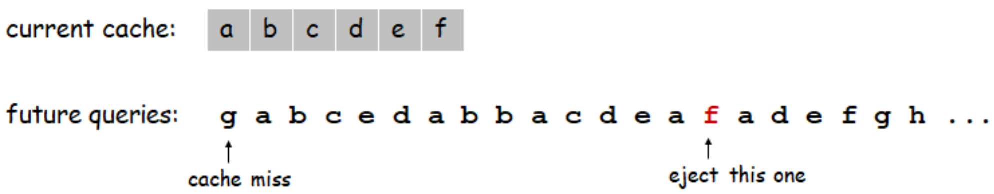
需求：a, b, c, b, c, a, a, b

有 2 次缓存未命中



6.2 算法

- [Farthest-in-future]-[FF]：清除缓存中直到未来最远时间才请求的项

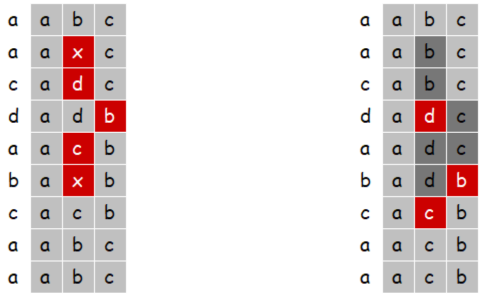


6.3 定义

简化的方案

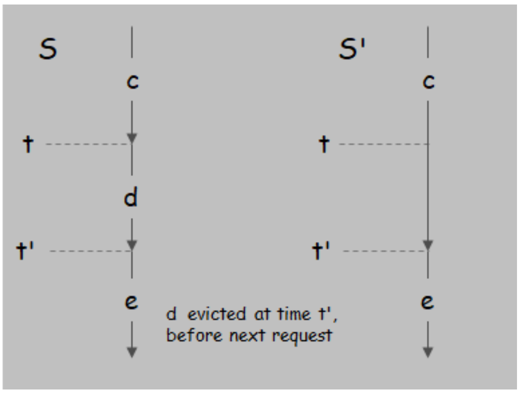
如果一个方案中，我们只有在需要的时候才把它加入缓存中，则我们称这个方案是简化的

对于一个未简化的方案，我们可以把它转化为一个简化的方案，而不增加剔除的次数

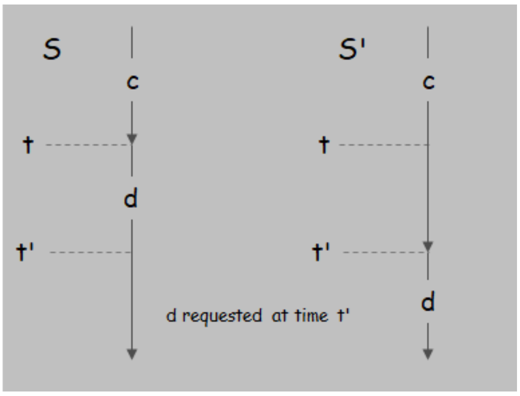


an unreduced schedule

- 假设方案 S 在时刻 t 把 d 加入缓存，而当前没有对 d 的访问需求
- 令 c S 中同时是被剔除的元素，可以在 t 时刻不做 c 和 d 的交换
- 设 c 为 S 将 d 带入缓存时要清除的项
  - 若 t' 时候， d 被剔除且换成了 e ，那么不需要在之前剔除 c



  - 若 t' 时候， d 需要被读入缓存，那么可以在这个时候再剔除 c



- 剔除的次数没有增加，但是方案更加简化

6.4 应用

在线算法和离线算法

- 离线 Offline：请求的完整序列是已知的
- 在线 Online：请求并不事先知道

缓存是CS中最基本的在线问题之一

贪婪算法方案

- [LIFO]：剔除最近被加入的元素
- [LRU]：剔除上一次访问时间最早的元素

定理

对于离线情况，FF 是最优算法

对于在线情况

- LIFO 是任意坏的
- LRU 是 k-竞争的

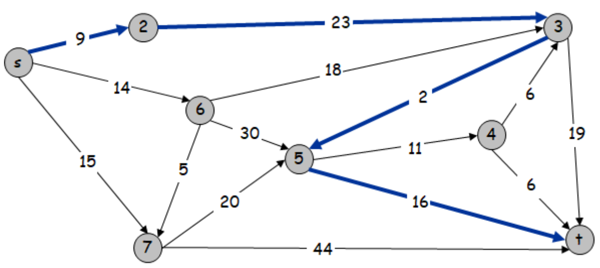
7. 图的最短路径 Shortest Paths in a Graph

7.1 问题描述

给定有向图 $G=(V,E)$，源点 s 和终点 t

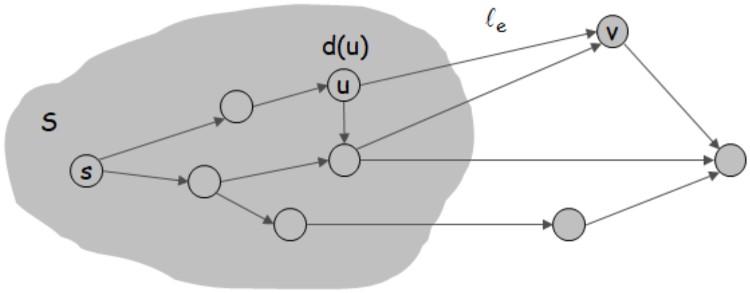
对于每条边 e，有一个长度 $l_e \geq 0$

求从 s 到 t 的最短路径 $d(t)$



- 如上图，最短路径蓝色的总长度为 48

7.2 算法 Dijkstra

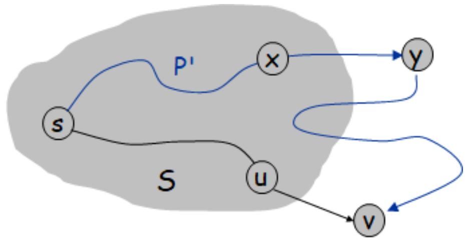


- 距离 $dist[\,]$，$dist[s]=0$，其它设为无穷
- 令 S 是一个探索过的节点的集合，初始的时候将s 放入集合，$S=\{s\}$
- 重复下面操作直到集合为空
  - 从 S 中找到 $dist[\,]$ 最小的节点 u
  - 对于 u 的每一条出边， $e=(u,v)$，计算
  - $dist[v]=mindist[v],dist[u]+l_e$
  - 如果 v 没有访问过，将 v 放入 S 中
  - 将 u 从 S 中移出

7.3 证明-归纳法

对于所有的点 u，Dijkstra 算法后的 $dist[u]$ 就是从 s 到 u 的最短路径

- 若只有一个节点，那么情况成立
- 假设有 k 个节点，Dijkstra 算法是正确的，目前已经访问过的集合为 S
- 令 $v=k+1$ 是下一个被加入的节点，(u,v) 是算法所选择的边
- 根据归纳假设，$s-u$ 的路径加上 (u,v) 是一条 $s-v$ 的路径长度，设为 $\pi(v)$
- 考虑任何一条从 $s-v$ 的路径 P，证明它的长度 $|P|\geq\pi(v)$



- 令 $x-y$ 是 P 离开 S 集合的第一条边，y 可以到达 v，让 P' 是 S 内到 x 的一条路径，有 $l(P)\geq l(P')+l(x,y)\geq d(x)+l(x,y)\geq\pi(y)$
- 由于 u 比 v 更早选中，则 $\pi(y)\geq\pi(v)$
- 故归纳假设成立
- 故得证

7.4 时间复杂度分析

时间复杂度为 $O((m+n)logn)$

根据算法的优化程度不一样，时间复杂度不同

8. 最小生成树 Minimum Spanning Tree

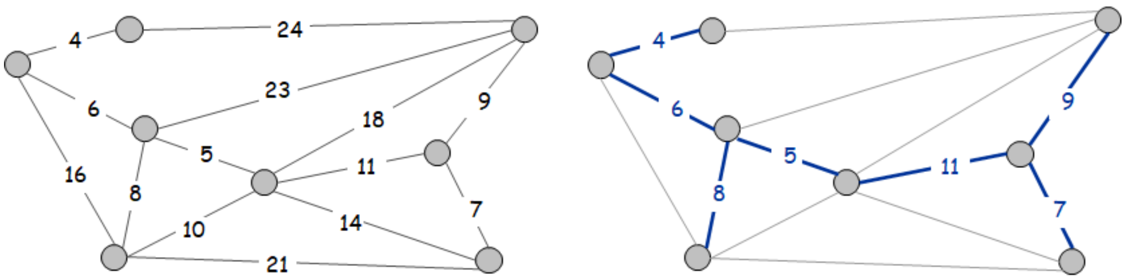
8.1 问题描述

给定一个连通图 $G=(V,E)$，令边的权重为 c_e

包含所有节点的树，我们称为生成树 spanning tree

支撑树中，所有边的总权重最小的数称为最小生成树，简称为 MST

目标是找到这样的 MST



$G = (V, E)$　　　$T, \ \Sigma_{e \in T} c_e = 50$

- 选择蓝色的边，我们得到一个总权重为 50 的最小支撑树

8.2 算法

Kruskal 算法

令 T 是空集，把边按照权重升序排列，对于当前的边，如果和 T 中的边不形成环的话，就把它加入 T，当加入的边能连接所有节点时，算法结束

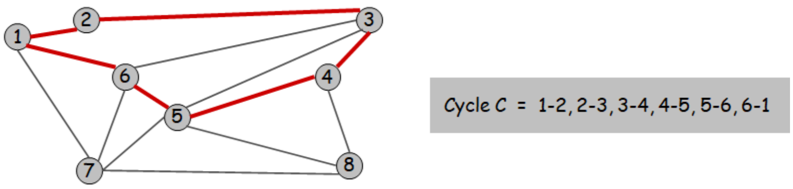
Prim 算法

令某一个节点 s 为根节点，从 s 出发贪婪地生成树 T，每一次从在恰好与 T 中叶节点相连的边里选取权重最小的边加入 T，直到 T 中包含所有节点

8.3 概念定义

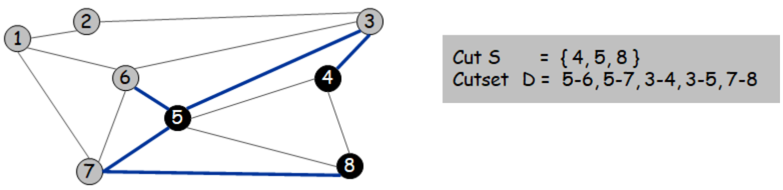
环 Cycle

首尾相连的边的集合



割 Cutset

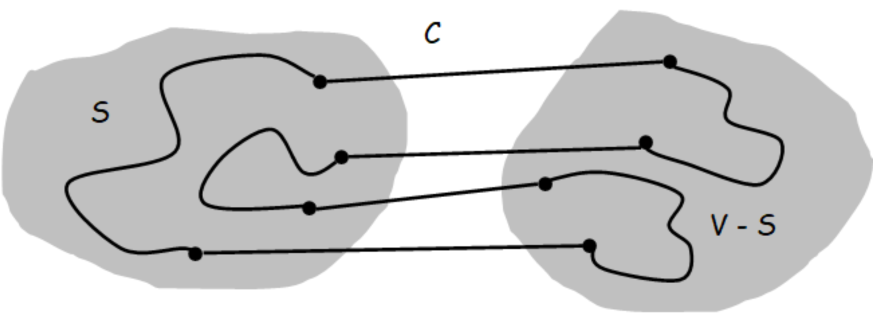
是点集合 S 中的一个子集，其中所有恰好有一个端点在 S 中的边构成的一个集合，称为 S 对应的割集



8.4 性质

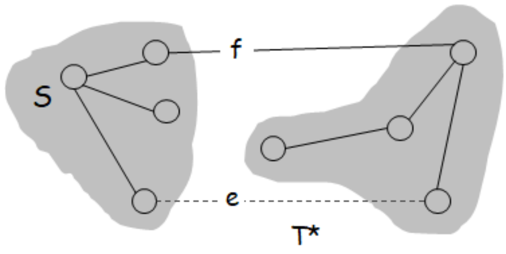
环和割的交集一定包含偶数条边

给定环 C 和割 S，环 C 中的边离开集合 S 和进入集合 S 的次数是相等的，所以 C 和 S 对应的交集中一定包含偶数条边



8.5 证明

假设所有边的权重是不同的



割最优性质

令 S 是任意一个点集的子集，恰有一个端点属于 S 的边中，开销最小记为 e ,则最小生成树一定包含 e

- 假设最小生成树不包含 e，我们把 e 加入 T 中，它会生成唯一一个环 C，且在割集合 D 中，那么一定存在另一条边 f， 它也在 C, D 中
- 如果我们删去 f 加入 e ，那么得到最小生成树 T'， 因为 $c_e < c_f$ 所以 $cost(T') < cost(T)$

环最优性质

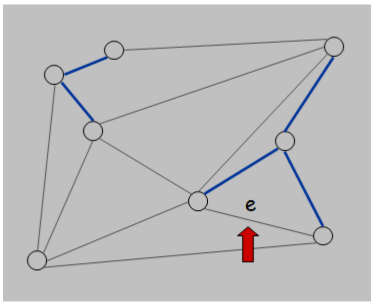
令 C 是一个环， f 是这个环上开销最大的边，则最小生成树一定不包含 f

- 假设 f 属于 $T*$，把它从 $T*$ 中删除，我们会得到一个割 S，边 f 同时在 C, S 中，那么一定存在另一条边 e 也在 C, S 中
- 我们删除 f 加入 e，也得到一个最小生成树 T'， 因为$c_e < c_f$，所以 $cost(T*) > cost(T)$，这与 $T*$ 是最小生成树矛盾

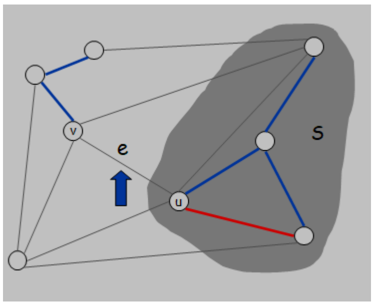
Krukal 算法的正确性

- 考虑边按权重升序排列

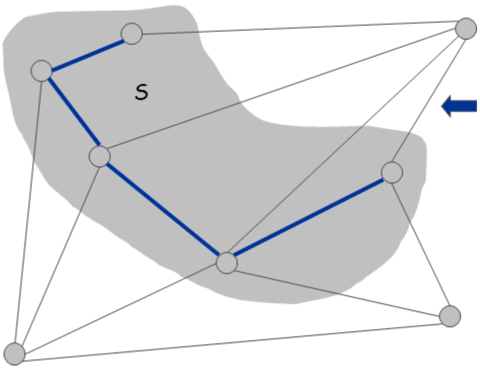
- 如果将 e 加到 T 中会产生一个环，e 一定是这个环上权重最大的边，根据环最优性质，舍弃



- 否则，插入 $e=(u,v)$ 进入 T，e 一定在某一个割集 S 中是费用最小的边，满足割最优性质



Prim 算法的正确性



- 初始化 S 为任意节点
- Prim 算法的思路满足割最优性质
- 将 S 对应的割集中的最小成本边添加到 T，并向 S 添加一个新的探索节点 u

8.6 时间复杂度分析

时间复杂度均为 $O(m\ logn)$

8.7 应用

MST是不同应用程序的基本问题

网络设计

- 电话、电气、液压、电视电缆、计算机、道路

NP-Hard 问题的近似算法

- 旅行商问题，斯坦纳树

聚类分析

9. 聚类 Clustering

9.1 问题描述

给定一个集合 U 中，有 n 个对象，标记为 $p_1, \ldots, p_n$，把它们分成相关的一些簇

划分成簇，以便不同簇中的点相距很远

将对象划分为 k 个非空簇

距离函数：指定两个对象的“远近程度”的数值

给定个体上的距离函数，聚类问题想要对它们进行分组，目标是使得同一组的个体是“接近的”，而不同组的个体是“远离的”

9.2 概念定义

距离 distance

给定集合 U，其个体标记为 $p_1, p_2, \ldots, p_n$，每个个体 p_i和 p_j，有一个数值距离 $d(p_i, p_j)$

- 两个对象之间的距离 $d(p_i, p_j) = 0$ 当且仅当 $p_i = p_j$
- $d(p_i, p_j) \geq 0$
- $d(p_i, p_j) = d(p_j, p_i)$

目标

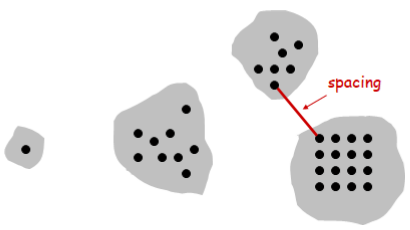
- 对于给定参数 k，我们寻找将 U 中的个体划分成 k 组， U 是 k 聚类的，即把 U 分成 k 个非空集合 $C_1, C_2, \ldots, C_k$ 的划分

k 聚类的间隔 spacing

- 处在不同聚类中的任何一对点之间的最小距离

优化目标：聚类-类的间隔最大

- 给定一个整数 k，找到 k 个类，使得 k 个类的间隔最大



9.3 算法

我们把对象对应为图中的节点，并把每个节点作为一个类，得到了 n 个类

寻找来自不同类中最近的两个点，并把它们连接起来，此时有 $n-1$ 个类

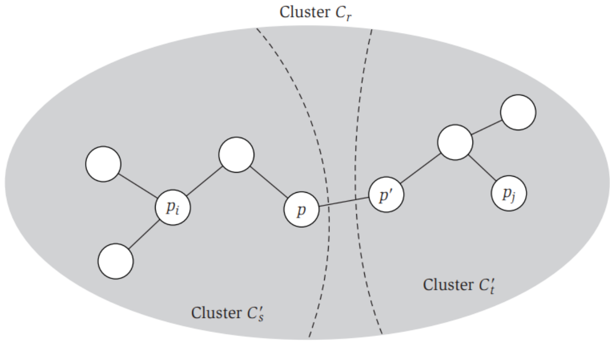
重复上述操作，直到恰有 $n-k$ 个类

等价于 Kruskal 算法

- 或者是在生成最小生成树后删除 $k-1$ 条费用开销最大边

9.4 证明-反证法

由删除最小生成树 T 的 $k-1$ 条最贵的边所构成的连通分支 $C_1, C_2, \ldots, C_k$ 组成一个最大间隔的 k 聚类



1. 设 C 表示聚类 $C_1, C_2, \ldots, C_k$， C 的间隔正好是最小生成树中第 $k-1$ 条最贵的边的长度 $d*$ ，这就是使用 Kruskal算法后的结果
2. 考虑某个其他的 k 聚类 C'，它把 U 划分成非空的集合 $C'_1, C'_2, \ldots, C'_k$ 我们必须证明 C' 的间隔至多为 $d*$
3. 因为两个聚类 C 和 C' 不相同，C 中一定存在某个聚类 C_r 不是 C' 中的某一个聚类 C'_s ，因此存在点 $p_i, p_j \in C_r$ 属于 C' 中的不同聚类
4. 由于 p_i, p_j 属于同一个连通分支 C_r，它一定是我们停止 Kruskal 算法添加了 p_i, p_j 路径 P 上的所有的边，特别的，这意味着 P 上的每条边长度至多是 $d*$
5. 现在我们知道 $p_i \in C'_s$，但是 $p_j \notin C'_s$，所以令 p' 是在 P 上第一个不属于 C'_s 的节点，令 p 是在 P 上紧接着 p' 之前的节点，我们刚刚论证了 $d(p,p') \leq d*$，但是 p 和 p' 属于聚类 C' 中的不同集合，因此 C' 的间隔至多是 $d(p,p') \leq d*$ ，不满足聚类的优化目标
6. 故得证

9.5 应用

- 移动自组织网络中的路由
- 识别基因表达模式
- 网页搜索的文档分类
- 医学图像数据库的相似性检索