研究生算法课课堂笔记

上课日期：2016 年 12 月 19 日　　　　　第(1)节课

组长学号及姓名：1601214428 张茜

组员学号及姓名：1601214445 朱碧莹

1601214425 班义琨

内容概要

本节课主要讲述了最小生成树的相关算法（Prim, Kruskal），以及算法设计上的关于贪心法的课后习题。内容如下：

1) 最小生成树的相关知识及 Prim 算法
2) Kruskal 算法
3) 贪心法课后习题 14

详细内容

1. 最小生成树的相关知识及 Prim 算法

1） 环（cycle）

环是指一条除了路径起点和终点之外，没有其他重复节点的路径。

Example 1：

如图 1.1 所示，1-2，2-3，3-4，4-5，5-6，6-1 就是一个环。

2） 割（cut）和割集（cutset）

割是指把一个图划分为两个非空子集 S 和 V-S，S 的割集是指只有一个顶点在 S 中的所有边的集合。

Example 2：

图 1.2 中的割 Cut S ＝｛ 4，5，8 ｝

S 的割集 Cutset D ＝｛ 5-6，5-7，3-4，3-5，7-8 ｝

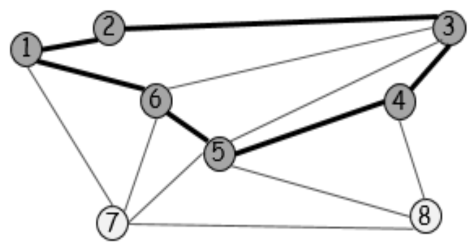


图 1.1

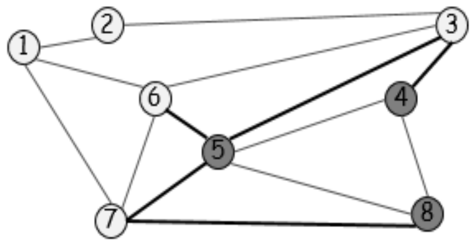


图 1.2

推论：环与割集相交的集中一定有偶数条边。

3) 生成树（ Spanning Tree ）

如果无向图 G 的一个子图是一棵包含 G 的所有顶点的树，则该子图称为 G 的生成树。生成树是连通图的包含图中的所有顶点的极小连通子图。图的生成树不唯一。从不同的顶点出发进行遍历，可以得到不同的生成树。所有生成树中所有边的权重之和最小的为最小生成树。

4) Blue rule（cut property）

如图 1.3，如果图中所有的边的权重都不同，那么：

假设 S 是一个割，e 是 S 的割集中权重最小的边，那么 e 一定在最小生成树中。

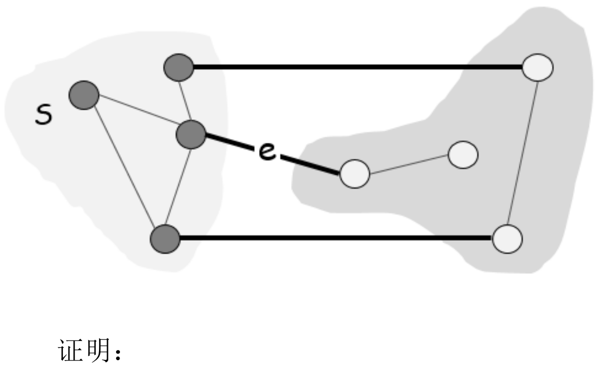


图 1.3

证明：

如图 1.4，假设 e 不属于最小生成树 T*，那么加上 e 就形成了一个环，那么 e 一定就在 S 的割集中，同样的，也一定有另一条边 f，也在 S 的割集中。因此，T' = T* ∪ {e} - {f}也一定是一棵生成树。

根据假设，e 是权重最小的边，因此一定有 Ce<Cf，所以，一定有 cost(T') < cost(T*)。显然和我们的假设是矛盾的，因此 e 一定在最小生成树中。

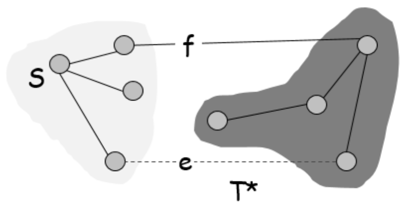


图 1.4

5) **Red rule （cycle property）**

如图 1.5 所示，如果图中所有的边的权重都不同，那么：

假设 C 是图 G 中的一个环，f 是环中权重最大的边，那么，最小生成树中一定不包含边 f。

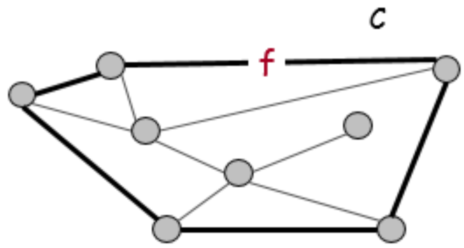


图 1.5

证明：

同理，如图 1.6，我们假设最小生成树 T*中是包含 f 的，环中另一条边 e 被删除了。但是，如果加入 e 删除 f，会生成一棵新的树 T'=T* ∪ {e} - {f}，也是一棵生成树。因为 f 是权重最大的边，因此 Ce<Cf，则有 cost(T') < cost(T*)，与假设矛盾。因此最小生成树中必然不包含 f。

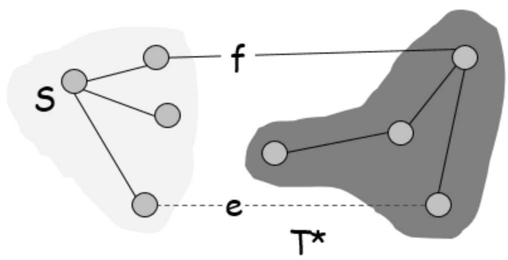


图 1.6

6) Prim 算法

Prim 算法是 blue rule 的应用，和 Dijkstra 算法类似，区别在于 Dijkstra 算法求的是从某一个点出发到各个顶点距离的最小值，需要指定开始顶点，而 Prim 算法可以从任意顶点开始。Prim 算法的时间复杂度是 O(m log n)

算法流程图如图 1.7 所示：

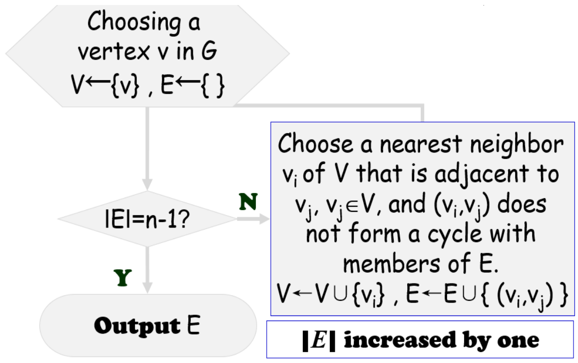


图 1.7

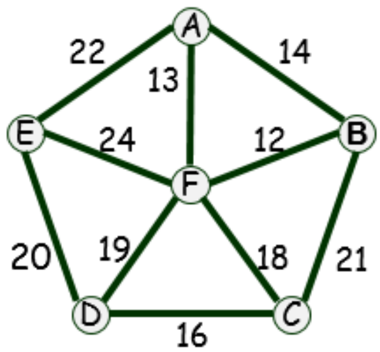


图 1.8

为了更好的理解 prim 算法，这里给出一个具体的例子（图 1.8）来对 prim 算法的流程进行展示：

假设有一个图 G，图中有 A, B, C, D, E, F 六个顶点，各个顶点之间的边及权重如图所示，选择 A 点作为起点，用 prim 算法来生成一棵最小生成树的步骤如下表 1.1 所示

S	A	B	C	D	E	F
{}	0/N	∞/N	∞/N	∞/N	∞/N	∞/N
A		14/A	∞/N	∞/N	22/A	13/A
A,F		12/F	18/F	19/F	22/A	
A,F,B			18/F	19/F	22/A	
A,F,B,C				16/C	22/A	
A,F,B,C,D					20/D	
A,F,B,C,D,E						

表 1.1

2. Kruskal 算法

1）算法流程：

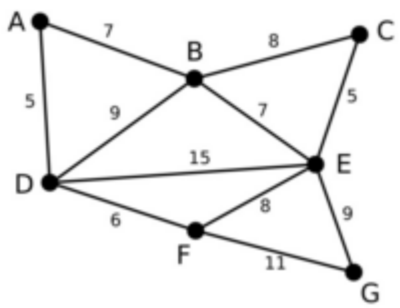
1.将图各边按照权值进行排序

2.将图遍历一次,找出权值最小的边,（条件：此次找出的边不能和已加入最小生成树集合的边构成环),若符合条件，则加入最小生成树的集合中。不符合条件则继续遍历图，寻找下一个最小权值的边。

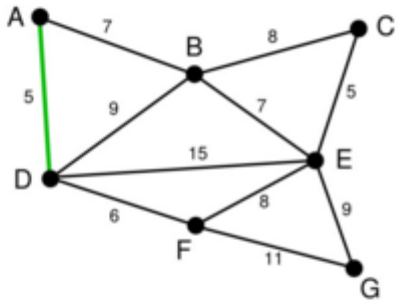
3.递归重复步骤 1，直到找出 n-1 条边为止（设图有 n 个结点，则最小生成树的边数应为 n-1 条),算法结束。得到的就是此图的最小生成树。

图例描述：

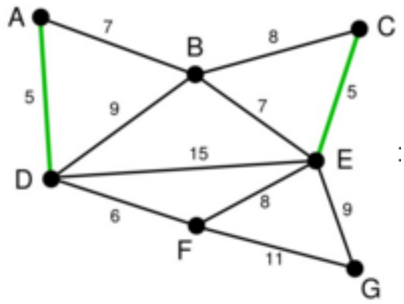
1. 首先第一步，我们有一张图 Graph，有若干点和边



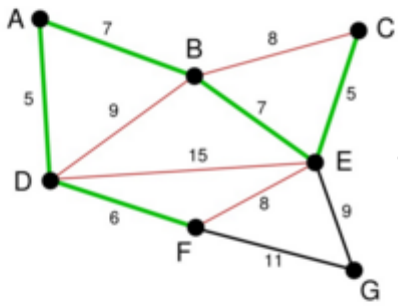
2. 将所有的边按长度排序，用排序的结果作为我们选择边的依据。这里再次体现了贪心算法的思想。排序完成后，我们率先选择了边 AD。这样我们的图就变成了下图



3. 在剩下的边中寻找，我们找到了 CE，边的权重也为 5

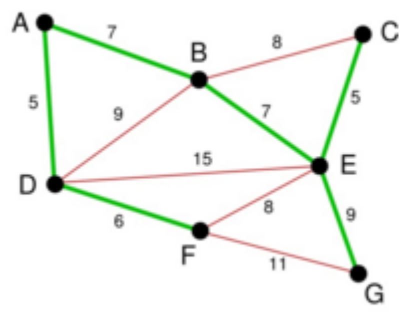


4. 依次类推我们找到了 6,7,7，即 DF，AB，BE



5. 下面继续选择， BC 或者 EF 尽管现在长度为 8 的边是最小的未选择的边。但是现在他们已经连通了（对于 BC 可以通过 CE,EB 来连接，类似的 EF 可以通过 EB,BA,AD,DF 来接连）。所以不需要选择他们。类似的 BD 也已经连通了（这里上图的连通线用红色表示了）。

最后就剩下 EG 和 FG 了。当然我们选择了 EG。最后成功的图如下所示：



2）**定理 Kruskal 算法产生 G 的一颗最小生成树**

证明：考虑任意一条由 Kruskal 算法加入的边 e=(v,w)，就在 e 加入之前的那一刻从 v 有路径通向某些结点，令 S 是所有这种结点的集合。显然 v∈S，但是 w∉S，因为加入 e 以后不会构成圈。此外，还没有遇到从 S 到 V-S 的边，因为如果任何这样的边可以被加入并且没构成圈，那么它只能是 Kruskal 算法加入的边。于是 e 是一个端点在 S 中而另一个端点在 V-S 中的最便宜的边，根据割性质，它属于每一颗最小生成树。证毕。

3）**算法实现：**

考虑：

1. 将图各边按照权值进行排序

2. 在排序操作之后，加入边的时候，我们使用 Union-Find 数据结构来维护（V,T）的连通分支。在每条边 e=(u,v)被考虑时，我们计算 Find(u)和 Find(v)并且检查它们是否相等以确定 u 与 v 是否属于不同的连通分支。如果算法决定把边 e 包含在树 T 中，我们使用 Union（Find(u)，Find(v)）来合并这两个分量。

```
KRUSKAL (V, E, c)
_____________________________________________
SORT m edges by weight so that c(e1) ≤ c(e2) ≤ ... ≤ c(em)
S ← φ
FOREACH v ∈ V:  MAKESET(v).
FOR i = 1 TO m
   (u, v) ← ei
   IF FINDSET(u) ≠ FINDSET(v)      ← are u and v in same component?
      S ← S ∪ { ei }
      UNION(u, v).      ← make u and v in same component
RETURN S
_____________________________________________
```

4) **算法分析：**

在算法的整个进程中，我们做了总计至多 2m 次 Find 操作和 n-1 次 Union 操作，而排序的时间复杂度为 O(m log m)

5）**Reverse-delete 算法**

简单描述:

该算法基于 Red-Rule，基本思想是

1. 将图各边按照权值进行排序
2. 在图中找环，并删除环中权值最大的边
3. 重复步骤 2，直至图中无环。

分析:

该算法的复杂度很高，实际中很少用，只有在特殊情况下才会用该算法。

3. 贪心法课后习题 14

1) 问题描述

有一个安全部门，存在 n 个进程 M <m_1,m_2,m_3…m_n> ，每个进程 m_i 有起始时间 start 和终止时间 finish。安全部门有一个检测程序 status-check，每启动一次可以检查到该时间点正在运行的进程，现在有 n 个进程 m_i，问最少启动多少次 status-check 可以把所有的 n 个进程都检查一遍。

2) 题目分析

首先最显然的办法就是每个进程都运行一次检测程序，可以把每个进程都检查一次，那么要启动 n 次。但是假如说有两个进程 m_1,m_2 的运行时间有交集，那么只需运行一次 status-check 就可以把这两个进程 m_1. m_2 检查。如图 3. 1 所示：

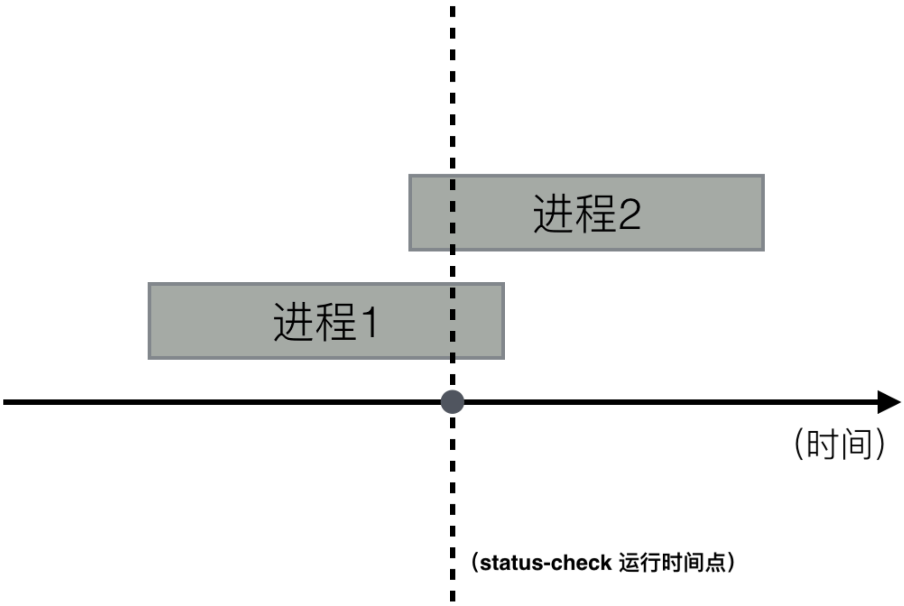


图 3. 1 status-check 运行时间点

3) 解题方案

这道题的本质是点对区间（每个进程 m_i 的 start 到 finish）的覆盖，假设有 n 个区间，要用最少的检测时间点对 n 个区间进行覆盖，即每个区间 m_i 至少含有一个检测时间点。那么本题的问题是：最少可以用多少个 status-check 运行时间点，能把 n 个进程区间都全部覆盖。

Observation :在 N 个区间中, 有 k 个两两都不冲突的区间, 那么在选择检查点时, 至少要有 k 个检查点。

证明：因为每个区间至少要有 1 个检查点，那么 k 个不相交的区间至少要有 k 个检查点才能满足对这 k 个区间的覆盖。

所以得出，在 n 个进程中，找到 k 个互不相交的区间，k 就是所需检查点的数目。

Question 1：在 n 个区间里，选取 K 个互不相交的区间，应该如何选？

Answer 1：其方法就是之前讲过的活动选择问题（interval scheduling）。首先按照贪心法则，以结束时间排序，按照结束时间从早到晚放入集合 S，且保证每次放入时，不与 S 中最晚结束的区间冲突。最终集合 S 中的区间就为 k 个互不相交的区间。如图 3.2 所示（详见笔记 11 月 24 日第二节）。

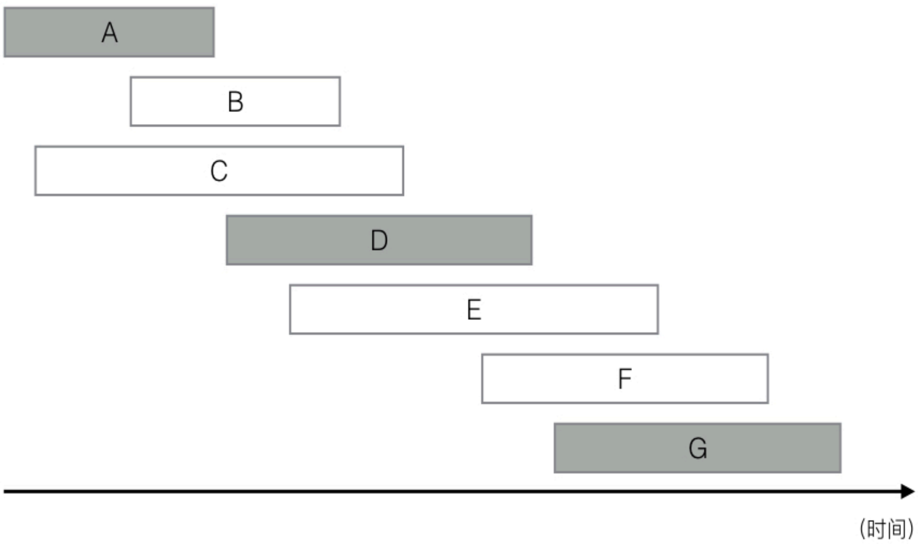


图 3.2 区间选择

算法描述如下：

EARLIEST-FINISH-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT jobs by finish time so that $f_1 \le f_2 \le \ldots \le f_n$

$A \leftarrow \emptyset$ (set of jobs selected)

FOR j = 1 TO n

  IF job j is compatible with A

    $A \leftarrow A \cup \{j\}$

RETURN A

Question 2：当求出了 k 个不相交的区间，为保证覆盖 n 个区间，那么应该如何选择时间点呢？

Answer 2：互不相交的 k 个区间为集合 S $<s_1,s_2,s_3\ldots s_k>$，存在与 S 中区间 s_i 冲突的区间为集合 T$<t_1,t_2,t_3\ldots t_{n-k}>$。在选择互不相交区间的过程中，按照贪心法则，以结束时间依次把 m_i 尝试放入 S 中，若区间 m_i 与 S 中最后一个放进去的区间 s_j 不存在冲突，放入

集合 S，若存在冲突，则所以放入集合 T。所以 T 中的区间 t_i 一定与其前一个放进 S 中的区间 s_j 冲突，即一定与 s_j 的结束时间点冲突。如图 3.3 所示。

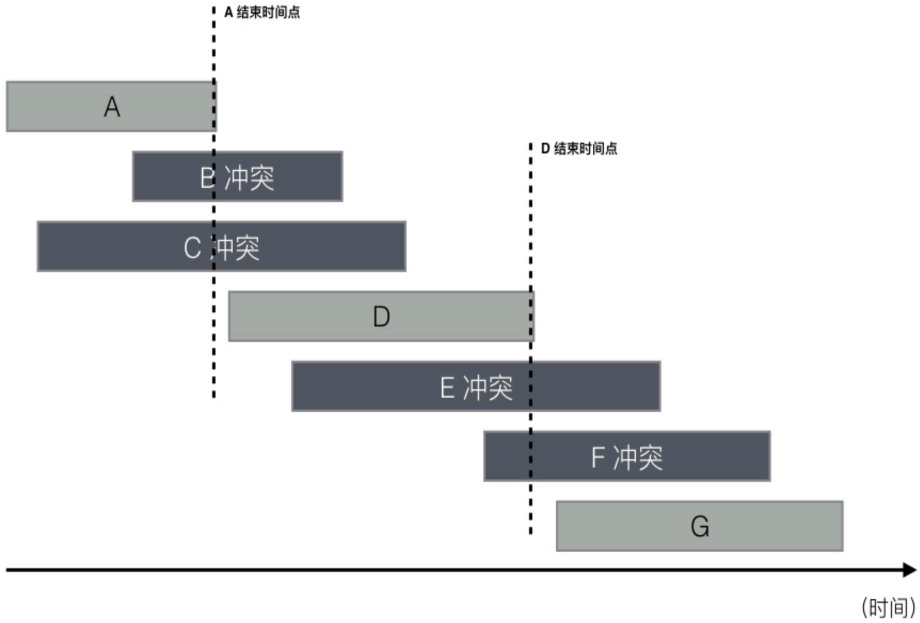


图 3.3 冲突时间点

所以以不相交区间集 S 中的 s_i 的结束时间点为检测点，一定可以覆盖 S 和 T 中所有区间，即覆盖所有进程。

同样也可以把区间选择问题转化为**最大独立集**问题。

把每个进程设为一个顶点 v，若任意两个进程运行的时间区间存在冲突，则该两个顶点间存在一条边 e。据此可以做出一个区间图（Interval Graph）G。对于图 G=(V,E)，V 是顶点集合，E 是边的集合。G 中的最大独立集就是两两互不冲突的区间集 S。如图 3.4 所示

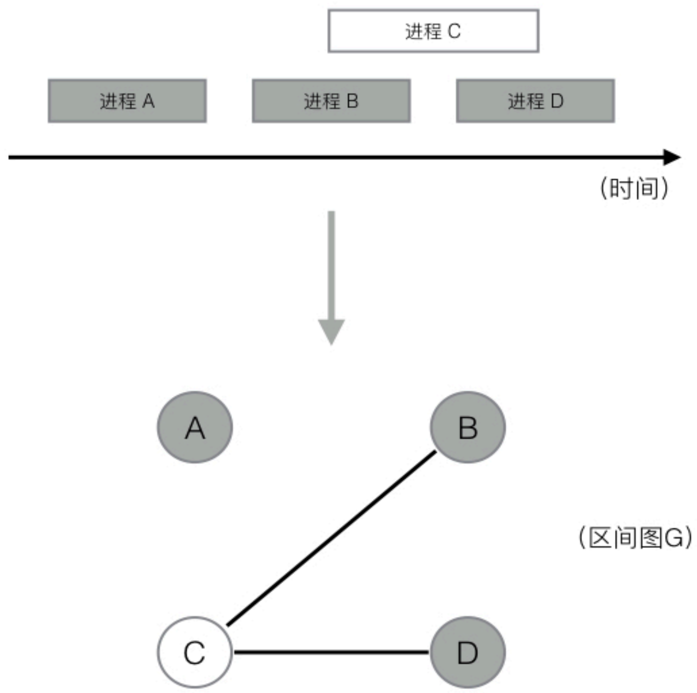


图 3.4　转化为最大独立集

4) 综上所述

此题的关键在于按照贪心法则，以区间的结束时间排序，选出 k 个两两互不相交的区间集 S，并以 S 中每个区间结束时间点为检查点，便可以达到对所有区间的覆盖。