算法分析与设计: 第七次作业

PB20061372　朱云沁　May 30, 2023

16.1-2 假定我们不再一直选择最早结束的活动, 而是选择最晚开始的活动, 前提仍然是与之前选出的所有活动均兼容.

(1) 请利用这一方法设计贪心算法和动态规划算法;

(2) 证明贪心算法会产生最优解.

解: 假设集合 $S = \{a_1, a_2, \dots, a_n\}$ 中每个活动 a_i 的开始时间为 s_i, 结束时间为 f_i, 且活动按开始时间递增排序, 即 $s_1 \le s_2 \le \cdots \le s_n$. 记 S_{ij} 为在 a_i 结束之后开始, 在 a_j 开始之前结束的活动子集.

- 最优子结构:

  令 A_{ij} 为 S_{ij} 的一个最大兼容活动子集, 包含活动 a_k. 下证 $A_{ik} = A_{ij} \cap S_{ik}$ 和 $A_{kj} = A_{ij} \cap S_{kj}$ 分别为子问题 S_{ik} 和 S_{kj} 的最大兼容活动子集.

  假设 A_{ik} 不是 S_{ik} 的最大兼容活动子集, 则存在 A'_{ik} 为 S_{ik} 的最大兼容活动子集, 且 $|A'_{ik}| > |A_{ik}|$. 由于 A'_{ik} 中的活动在 a_k 开始之前结束, 故 $A'_{ik} \cup \{a_k\} \cup A_{ij}$ 为 S_{ij} 的兼容活动子集, 而 $|A'_{ik} \cup \{a_k\} \cup A_{ij}| > |A_{ij}|$, 与 A_{ij} 为 S_{ij} 的最大兼容活动子集矛盾. 因此 A_{ik} 为 S_{ik} 的最大兼容活动子集. 同理可证 A_{kj} 为 S_{kj} 的最大兼容活动子集.

- 递归式:

  令 $c[i,j]$ 为 S_{ij} 的最大兼容活动子集的规模, 则有

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \ne \emptyset \end{cases}$$

- 动态规划算法:

  构造最优解 $A_{0,n+1}$ 的过程从略, 详见 OJ 代码.

Algorithm 1: 16.1-2 (dynamic programming)

Input: arrays s and f of length n, where s_i is the start time of activity a_i and f_i is the finish time of activity a_i for $i = 1, 2, \dots, n$.
Output: the size of a maximum set of mutually compatible activities.

```
Sort the activities in non-decreasing order of their start times.
$f_0 \leftarrow -\infty$
$s_{n+1} \leftarrow \infty$
Initialize $c[i,j] = 0$ for $i = 0, 1, 2, \dots, n$ and $j = 1, 2, \dots, n, n+1$.
for $l \leftarrow 2$ to $n+1$
    for $i \leftarrow 0$ to $n+1-l$
        $j = i + l$
        for $k \leftarrow i+1$ to $j-1$
            if $s_k \ge f_i$ and $f_k \le s_j$
                $c[i,j] = \max\{c[i,j], c[i,k] + c[k,j] + 1\}$
return $c[0, n+1]$
```

- 贪心选择:

  记 S_k 为在 a_k 开始之前结束的活动集合, 且 a_m 为 S_k 中开始时间最晚的活动, 则选择 a_m 后, S_m 为唯一非空的子问题. 下证 a_m 在 S_k 的某个最大兼容活动子集中.

  令 A_k 为 S_k 的一个最大兼容活动子集, 且 a_j 为 A_k 中开始时间最晚的活动. 若 $a_j = a_m$, 则 a_m 在 A_k 中. 否则, 由于 a_m 的开始时间不早于 a_j, 故 a_m 与集合 $A_k \setminus \{a_j\}$ 中的活动不相交, 进而 $A'_k = A_k \setminus \{a_j\} \cup \{a_m\}$ 为 S_k 的一个兼容活动子集. 又由于 $|A'_k| = |A_k|$, 故 A'_k 为 S_k 的一个包含 a_m 的最大兼容活动子集.

- 贪心算法:

Algorithm 2: 16.1-2 (greedy)

Input: arrays s and f of length n, where s_i is the start time of activity a_i and f_i is the finish time of activity a_i for $i = 1, 2, \dots, n$.
Output: the size of a maximum set of mutually compatible activities.

```
Sort the activities in non-decreasing order of their start times.
$A = \{a_n\}$
$k = n$
for $m = n-1, n-2, \dots, 1$
    if $f_m \le s_k$
        $A = A \cup \{a_m\}$
        $k = m$
return $|A|$
```

16.1-5 考虑活动选择问题的一个变形: 每个活动 a_i 除了开始和结束时间外, 还有一个值 v_i. 目标不再是求规模最大的兼容活动子集, 而是求值之和最大的兼容活动子集. 也就是说, 选择一个兼容活动子集 A, 使得 $\sum_{a_k \in A} v_k$ 最大化. 设计一个多项式时间的算法求解此问题.

解: 沿用 16.1-2 的记号, 将活动按开始时间递增排序.

- 最优子结构:

  令 A_{ij} 为 S_{ij} 的一个加权最大兼容活动子集, 包含活动 a_k. 下证 $A_{ik} = A_{ij} \cap S_{ik}$ 和 $A_{kj} = A_{ij} \cap S_{kj}$ 分别为子问题 S_{ik} 和 S_{kj} 的加权最大兼容活动子集.

  假设 A_{ik} 不是 S_{ik} 的加权最大兼容活动子集, 则存在 A'_{ik} 为 S_{ik} 的加权最大兼容活动子集, 且 $\sum_{a_j \in A'_{ik}} v_j > \sum_{a_j \in A_{ik}} v_j$. 由于 A'_{ik} 中的活动在 a_k 开始之前结束, 故 $A'_{ik} \cup \{a_k\} \cup A_{ij}$ 为 S_{ij} 的兼容活动子集, 而 $\sum_{a_j \in A'_{ik} v_j} + v_k + \sum_{a_j \in A_{ij}} v_j > \sum_{a_j \in A_{ij}} v_j$, 与 A_{ij} 为 S_{ij} 的加权最大兼容活动子集矛盾. 因此 A_{ik} 为 S_{ik} 的加权最大兼容活动子集. 同理可证 A_{kj} 为 S_{kj} 的加权最大兼容活动子集.

- 递归式:

  令 $c[i,j]$ 为 S_{ij} 的加权最大兼容活动子集的规模, 则有

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + v_k\} & \text{if } S_{ij} \ne \emptyset \end{cases}$$

- 动态规划算法:

  构造最优解 $A_{0,n+1}$ 的过程从略, 详见 OJ 代码.

Algorithm 3: 16.1-2 (dynamic programming)

Input: arrays s, f and v of length n, where s_i, f_i and v_i are the start time, finish time and value of activity a_i for $i = 1, 2, \dots, n$, respectively.
Output: the size of a maximum set of mutually compatible activities.

```
Sort the activities in non-decreasing order of their start times.
$f[0] \leftarrow -\infty$
$s[n+1] \leftarrow \infty$
Initialize $c[i,j] = 0$ for $i = 0, 1, 2, \dots, n$ and $j = 1, 2, \dots, n, n+1$.
for $l \leftarrow 2$ to $n+1$
    for $i \leftarrow 0$ to $n+1-l$
        $j = i + l$
        for $k \leftarrow i+1$ to $j-1$
            if $s_k \ge f_i$ and $f_k \le s_j$
                $c[i,j] = \max\{c[i,j], c[i,k] + c[k,j] + v_k\}$
return $c[0, n+1]$
```

- 时间复杂度:

  由于三重循环的迭代次数均为 $O(n)$, 故算法的时间复杂度为 $O(n^3)$, 满足多项式时间要求.

16.2-5 设计一个高效算法, 对实数线上给定的一个点集 $\{x_1, x_2, \dots, x_n\}$, 求一个单位长度闭区间的集合, 包含所有给定的点, 并要求此集合最小.

解: 假设集合 $X = \{x_1, x_2, \dots, x_n\}$ 满足 $x_1 \le x_2 \le \cdots \le x_n$. 记 $X_{ij} = \{x_i, x_{i+1}, \dots, x_j\}$, 并约定当 $i > j$ 时, $X_{ij} = \emptyset$.

- 最优子结构:

  令 S_{ij} 为覆盖 X_{ij} 的最小单位长度闭区间集, 其包含单位长度闭区间 $I = [\alpha, \alpha+1]$. 令 X_{kl} 为 X_{ij} 中被区间 I 覆盖的点的集合, 使得 $X_{kl} \subseteq I$ 且 $X_{i,k-1} \cap I = X_{l+1,j} \cap I = \emptyset$. 下证 $S_{i,k-1} = \{[x, x+1] \in S_{ij} \mid x < \alpha\}$ 和 $S_{l+1,j} = \{[x, x+1] \in S_{ij} \mid x > \alpha\}$ 分别为覆盖子问题 $X_{i,k-1}$ 和 $X_{l+1,j}$ 的最小单位长度闭区间集.

  假设 $S_{i,k-1}$ 不是覆盖 $X_{i,k-1}$ 的最小单位长度闭区间集, 则存在 $S'_{i,k-1}$ 为覆盖 $X_{i,k-1}$ 的最小单位长度闭区间集, 且 $|S'_{i,k-1}| < |S_{i,k-1}|$. 由于 $S'_{i,k-1}$, $\{I\}$, $S_{l+1,k}$ 分别覆盖 $X_{i,k-1}$, X_{kl}, $X_{l+1,j}$, 故 $S'_{i,k-1} \cup \{I\} \cup S_{l+1,j}$ 为覆盖 X_{ij} 的单位长度闭区间集, 而 $|S'_{i,k-1} \cup \{I\} \cup S_{l+1,j}| < |S_{ij}|$, 与 S_{ij} 为覆盖 X_{ij} 的最小单位长度闭区间集矛盾. 因此 $S_{i,k-1}$ 为覆盖子问题 $X_{i,k-1}$ 的最小单位长度闭区间集. 同理可证 $S_{l+1,j}$ 为覆盖子问题 $X_{l+1,j}$ 的最小单位长度闭区间集.

- 递归式:

  令 $c[i,j]$ 为覆盖 X_{ij} 的最小单位长度闭区间集的规模, 则有

$$c[i,j] = \begin{cases} 0 & \text{if } X_{ij} = \emptyset \\ \min\limits_{\substack{i \le k \le l \le j \\ x_l \le x_k + 1 < x_{l+1}}} \{c[i,k-1] + c[l+1,j] + 1\} & \text{if } X_{ij} \ne \emptyset \end{cases}$$

- 贪心选择:

  记 $X_k = \{x_k, x_{k+1}, \dots, x_n\}$. 若选择 x_k 作为闭区间的左端点, 且有 $x_l \le x_k + 1 < x_{l+1}$, 则 X_{l+1} 为唯一非空的子问题. 下证 $[x_k, x_k+1]$ 在 X_k 的某个最小单位长度闭区间集中.

  令 S_k 为覆盖 X_k 的最小单位长度闭区间集, 且 $I = [\alpha, \alpha+1]$ 为 S_k 中左端点最小的闭区间. 若 $x_k = \alpha$, 则 $[x_k, x_k+1] \in S_k$. 否则, 由于 $x_k > \alpha$, $[x_k, x_k+1]$ 覆盖的点的数目不少于 I 覆盖的点的数目, 故 $S'_k = S_k \setminus \{I\} \cup \{[x_k, x_k+1]\}$ 为覆盖 X_k 的单位长度闭区间集. 又由于 $|S'_k| = |S_k|$, 故 S'_k 为覆盖 X_k 的最小单位长度闭区间集.

- 贪心算法:

Algorithm 4: 16.2-5 (version 1)

Input: a point set $X = \{x_1, x_2, \dots, x_n\}$ of real numbers.
Output: a minimum set of unit-length closed intervals that contains all points in X.

```
Sort $X$ in non-decreasing order.
$S \leftarrow \emptyset$
$k = 1$
while $k \le n$
    $S = S \cup \{[x_i, x_i + 1]\}$
    // binary-search the smallest index $i$ such that $x_i > x_k + 1$
    $k = $ UPPER-BOUND$(X, k, n, x_k + 1)$
return $S$
```

Algorithm 5: 16.2-5 (version 2)

Input: a point set $X = \{x_1, x_2, \dots, x_n\}$ of real numbers.
Output: a minimum set of unit-length closed intervals that contains all points in X.

```
Sort $X$ in non-decreasing order.
$S \leftarrow \{[x_1, x_1 + 1]\}$
$k = 1$
for $i \leftarrow 2$ to $n$
    if $x_i > x_k + 1$
        $S = S \cup \{[x_i, x_i + 1]\}$
        $k = i$
return $S$
```

- 时间复杂度:

  除排序操作以外, 则算法 4 剩余部分的时间复杂度为 $O(|S| \log n)$, 算法 5 剩余部分的时间复杂度为 $O(n)$.

24.3-6 给定有向图 $G = \langle V, E \rangle$, 每条边 (u,v) 有一个关联值 $r(u,v)$, 该关联值是一个实数, 其范围是 $0 \le r(u,v) \le 1$, 代表的意思是从节点 u 到节点 v 之间的通信链路的可靠性. 可以认为, $r(u,v)$ 代表的是从结点 u 到结点 v 之间的通信链路不失效的概率, 并且假设这些概率之间相互独立. 请给出一个有效的算法来找到任何两个结点之间最可靠的通信链路.

解: 给定 G 上的一条路径 $p = \langle v_1, v_2, \dots, v_k \rangle$, 其对应通信链路的可靠性为 $\prod_{i=1}^{k-1} r(v_i, v_{i+1})$. 从 G 中删除关联值为 0 的边, 并定义 $d[u,v] = -\log r(u,v) \ge 0$ 为边 (u,v) 的权重. 最大化通信链路的可靠性等价于最小化路径的权重之和. 因此, 问题转化为求解 G 上任意两点间的最短路径, 可用 Dijkstra 算法求解. 最优子结构及贪心选择性质已在课本/课件中证明, 此处略去.

假设路径起点为 s, 终点为 t. 令 $T \subseteq V$ 为顶点的子集, $E^-(T)$ 为 T 中所有顶点的入边的集合. 记 $dis^-[x,T]$ 为子图 $G_T^- = \langle V, E^-(T) \rangle$ 中从 x 到 t 的最短路径长度. 则有

$$dis^-[x,T] = \min_{v \in T} \left\{ d[x,v] + dis^-[v, T \setminus \{v\}] \right\} \tag{1}$$

令 $S = V \setminus T$. 类似地, 记 $E^+(S)$ 为 S 中所有顶点的出边的集合, $dis^+[S,x]$ 为子图 $G_S^+ = \langle V, E^+(S) \rangle$ 中从 s 到 x 的最短路径长度. 初始状态下, $S = \{s\}$, $T = V \setminus \{s\}$. 每次递归计算 (1) 式, Dijkstra 算法贪心地选择 $v = \arg\min_{v \in T} dis^+[S,v]$, 将 v 从 T 中移入 S, 并按 (2) 式更新 $dis^+[S,x]$

$$dis^+[S,x] = \min \left\{ dis^+[S \setminus \{v\}, v] + d[v,x],\ dis^+[S \setminus \{v\}, x] \right\}, \forall x \in T \tag{2}$$

进而求解子问题. 伪代码如下:

Algorithm 6: 24.3-6

Input: a directed graph $G = \langle V, E \rangle$, where each edge $(u,v) \in E$ has a reliability $r(u,v)$; a source vertex s and a target vertex t.
Output: the most reliable path from s to t.

```
Initialize $d[u,v] = \infty$ for each $(u,v) \in V \times V$
for each $v \in V$
    $d[v,v] = 0$
for each $(u,v) \in E$
    if $r(u,v) > 0$
        $d[u,v] = -\log r(u,v)$
return SHORTEST-PATH$(G, d, s, t)$
```

```
function SHORTEST-PATH$(G, d, s, t)$
$S = \{s\}$
$T = V \setminus \{s\}$
for each $v \in T$
    $dis[v] = d[s,v]$
while $t \in T$
    $v = \arg\min_{v \in T} dis[S,v]$
    $S = S \cup \{v\}$
    $T = T \setminus \{v\}$
    for each $x \in T$
        $dis[x] = \min\{dis[x], dis[v] + d[v,x]\}$
return $dis[t]$
```

构造最优解的过程从略, 详见 OJ 代码.