# 第 11 章 在线算法

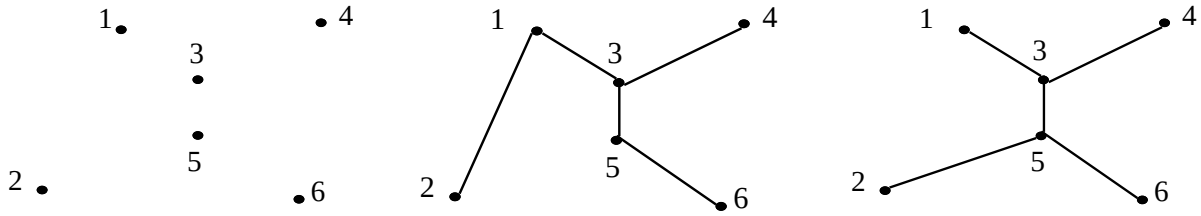
## 11.1 在线算法与竞争度分析

前面各章讨论的算法均假设算法开始运行之前，问题实例的所有输入数据均是已知的，因此算法运行时可以充分利用输入数据之间完整的关联特性，这类算法称之为**离线算法**。然而，许多实际应用中的计算问题却在求解该问题的算法运行之后，问题实例的数据才逐渐到来，算法必须在输入数据不是完全可知的情况下，完成相应的计算并输出计算结果，这类算法算法称之为**在线算法**。

例如，磁盘调度算法是在线算法，因为磁盘调度请求在调度算法开始运行之后才以完全无法预知的方式逐渐到来。类似地，操作系统的内存页面调度算法是在线算法，因为对内存页面的访问请求在调度算法运行之后才以完全无法预知的方式逐渐到来；再如，网络路由算法是在线算法，因为数据包的路由请求在路由算法开始运行之后才逐渐到来，每个数据包的目的地址无法预知。除了在计算机操作系统和网络路由领域中的应用之外，在线算法还在任务调度、数据结构、数据流处理、数据监测、知识在线学习、博弈过程和计算金融等领域具有十分广泛的应用。

在线算法均是近似算法。事实上，在线算法开始运行时，问题实例的输入数据不能完全可得，并且算法必须实时处理每个输入数据，因此在线算法的输出结果未必是问题的正确解或最优解。甚至，在线算法根据当前输入数据做出的决策可能使得算法的后续步骤必然无法获得问题的最优解。因此，在线算法设计过程中往往放弃“获取最优解”这一目标，改而求得问题的一个“较好的可行解”即可。

例如，考虑用在线算法在图11.1(a)所示的顺序到来的6个点上找出代价最小的生成树。在线算法每次处理新到达的顶点$i$时，均选择代价最小的边将顶点$i$连接到之前到达的顶点1,2,...,$i$-1上。由此，产生图11.1(b)所示的生成树，它显然不同于图11.1(c)所示的最小生成树。而且，如果还有后续顶点到来且后续顶点均位于1和5的连线的右侧，则在线算法得到的生成树必然不同于输入点集上的最小生成树。这是由于，在线算法输出的生成树必然使用边(2,1)将顶点2连接到其他顶点上，而最小生成树则必然使用边(2,5)将顶点2连接到其他顶点上。




(a)顺序到达的6个点　　(b)在线算法输出的生成树　　(c)输入点集的最小生成树

图11.1 在线算法在平面上顺序到达的6个点上产生的近似“最小”生成树

在线算法的时间复杂度的分析可以从两个方面进行。一方面，同离线算法一样，分析算法完成所有操作所需的总的时间复杂度。另一方面，还需要分析在线算法的新数据响应时间，即处理每个新到来的数据的时间复杂度。在一定程度上讲，在线算法

的新数据响应时间复杂度比算法的总时间复杂度更重要。这是由于，如果问题实例的输入数据密集地到来（相继两个输入数据的到达时间间隔很短），则在线算法的优劣主要取决于单步响应时间复杂度，而非总的时间复杂度；反之，如果问题实例的输入数据稀疏地到来（相继两个输入数据的到达时间间隔较长），则可以利用离线算法计算当前所有输入数据上的最优解，无需在线算法。因此，实际应用中，可以根据数据到达的密集程度和在线算法的新数据响应时间来选用在线算法。

在线算法还需要分析其竞争度。竞争度反映在线算法获得的解的质量好坏，这大致相当于近似算法的近似比。直观上，将在线算法和离线算法视为“竞争对手”，竞争度衡量在线算法获得的可行解相对于离线算法在同一问题实例的所有输入数据已知的情况下获得的最优解的优劣。

**定义11.1**.设问题$A$是一个输入数据在线到来的计算问题，算法$\mathcal{A}$是求解问题$A$的一个在线算法。对于问题$A$的任意实例$I_A$, 记$C_{on}(I_A)$是算法$\mathcal{A}$在实例$I_A$上获得的解的代价，$C_{off}(I_A)$表示最优离线算法在实例$I_A$获得的解的代价。如果$C_{on}(I_A)\le\alpha\cdot C_{off}(I_A)+\beta$对任意实例$I_A$成立，其中$\beta$是与输入规模$|I_A|$无关的常量，则称算法$\mathcal{A}$的**竞争度**为$\alpha$。如果问题$A$不存在其他竞争度小于$\alpha$的在线算法，则称在线算法$\mathcal{A}$是**最优的**。

由定义可以看出，在线算法的竞争度越接近于1，表明在线算法给出的近似解越接近最佳离线算法给出的解。分析在线算法的竞争度是在线算法分析的重点。由于在线算法的设计和竞争度分析往往交织在一起，因此在线算法的设计相对于离线算法的设计更难。

本章后续部分将给出几个在线算法设计和分析的过程，希望读者能够从中了解贪心策略、随机策略、排除策略、补偿策略、平衡策略在线算法设计中的应用，并理解组合优化方法、势能分析方法和反例方法在在线算法分析中的应用。11.2节讨论欧几里德平面上生成树的在线算法。11.3节讨论计算凸包的在线算法。11.4节讨论线性链表在线更新算法。11.5节讨论最短并行调度问题的在线算法。更多的内容请参考专著[8]。

## 11.2 欧几里德最小生成树问题的在线算法

欧几里德最小生成树问题的输入是欧几里德平面上的$n$个点，输出是这$n$个点上的最小生成树。在线欧几里德最小生成树问题中，输入点是逐个到来，并且要求每个输入点到来时，必须将它与之前构造的生成树相连，而且操作是不可逆的。

### 11.2.1 在线贪心算法

在线欧几里德最小生成树问题可以利用在线贪心策略求解。将第$k$个到来的顶点记为$v_k$。当顶点$v_k$到来时,算法选取$v_1,v_2,...,v_{k-1}$中距离$v_k$最近的顶点$v$，然后通过边$(v,v_k)$将$v_k$连接到现有生成树上。例如，在图11.1(a)所示依编号递增顺序到来的6个点上，算法将得到图11.1(b)所示的生成树，它不同于图11.1(c)所示的最小生成树。

该算法处理第$k$个到来的顶点的需要计算$k$-1次距离，故算法的新数据响应时间为$\Theta(k)$。算法处理规模为$n$的问题实例的需要计算$\sum_{k=2}^{n}(k-1)=\Theta(n^2)$次距离，故算法的时间复杂度为$\Theta(n^2)$。由于算法的新数据响应时间$\Theta(k)$随$k$递增而线性递增，该算法不是一个好的在线算法。

下面分析算法的竞争度。为此，将在线算法在规模为$n$的任意实例$I$上获得的生成树记为$T_{on}$,其代价记为$l_{on}$;将同一实例$I$上（由离线算法获得）的最小生成树记为$T^*$,其代

价记为$l$。注意，$T_{on}$恰含有$n$-1条边，$l_{on}$等于这$n$-1条边的长度之和。如果能建立$T_{on}$中每条边的长度与$l$之间的联系，则可获得竞争度。

**引理11.1**. 设$1\le k\le n$-1，则$T_{on}$中第$k$长的边的长度小于等于$2l/k$。

**证明**.记实例$I$中顺序到来的顶点依次为$v_1,v_2,...,v_n$。令集合

$$S_k=\{v_i|\text{在线算法处理}v_i\text{时在}T_{on}\text{中引入长度大于}2l/k\text{的边}\}$$

于是，引理成立当且仅当$|S_k|<k$。下面通过研究$S_k$的性质来证明$|S_k|<k$。

$S_k$中任意两个顶点间的距离大于$2l/k$。否则，设$v_i,v_j\in S_k$且$v_i$和$v_j$之间的距离小于$2l/k$。不妨设$i<j$,即$v_i$先于$v_j$到来。于是，在线算法处理$v_j$时，选取$(v_j,v_1),(v_j,v_2),...,(v_j,v_{j-1})$中最短的边将$v_j$连接到生成树，所选取的边必然不超过边$(v_j,v_i)$的长度。因此，在线算法处理$v_i$时在$T_{on}$中不会引入长度大于$2l/k$的边，这与$v_j\in S_k$矛盾。

于是，$S_k$上最短哈密顿环的代价至少为$\frac{2l}{k}\cdot|S_k|$。由于最短哈密顿环的代价不超过$S_k$上最小生成树代价的2倍（参见10.2.4节），故$S_k$的最小生成树的代价至少为$\frac{l}{k}\cdot|S_k|$。

由于$S_k\subseteq\{v_1,v_2,...,v_n\}$且距离满足（严格的）三角不等式，$S_k$的最小生成树的代价小于输入点集$\{v_1,v_2,...,v_n\}$的最小生成树$T^*$的代价$l$。于是 $\frac{l}{k}\cdot|S_k|<l$，亦即$|S_k|<k$。 □

由引理11.1可知，在线算法输出的生成树$T_{on}$的代价$l_{on}$可如下计算。

$$\begin{aligned} l_{on}&=\sum_{k=1}^{n-1}T_{on}\text{ 中第}k\text{长的边的长度}\\ &\le\sum_{k=1}^{n-1}\frac{2l}{k}\\ &\le 2l\cdot\ln n\end{aligned}$$

**定理11.2**. 求解欧几里德生成树问题的在线贪心算法的竞争度为$O(\log n)$。 □

## 11.2.2 在线随机算法

11.2.1节的分析表明,求解欧几里德生成树问题的在线贪心算法的一个显著缺陷是新数据响应时间$\Theta(k)$随$k$递增而线性递增。随着问题实例中输入数据规模的增长，贪心在线算法对新到来的数据响应越来越慢，对在线算法而言这通常是不可接受的。为了克服这一缺陷，引入参数$m$和随机选择策略来改进算法。

具体地讲，当顶点$v_k$到来时,如果$k\le m+1$,亦即当前到来的点的个数不超过$m+1$，则算法选取$v_1,v_2,...,v_{k-1}$中距离$v_k$最近的顶点$v$，然后通过边$(v,v_k)$将$v_k$连接到现有生成树上。否则，算法随机地从$v_1,v_2,...,v_{k-1}$中选取$m$个顶点，计算$v_k$与这些点间的距离，选择距离$v_k$最近的顶点$v$，然后通过边$(v,v_k)$将$v_k$连接到现有生成树上。算法的形式化描述如下。

**求解欧几里德生成树问题的在线随机算法OnlineRandMST($m$)**

**输入**：在线到来的顶点$v_1,v_2,...,v_n$,控制新数据响应时间的参数$m$

**输出**：$v_1,v_2,...,v_n$的一棵生成树$T$

1. Input($v_1$), Input($v_2$), $T\leftarrow\{(v_1,v_2)\}$;
2. For $k\leftarrow 3$ To $n$ Do
3. If $k\le m+1$ Then 添加$v_k$到$v_1,v_2,...,v_{k-1}$的最短边到$T$中;
4. Else 从$v_k$到$v_1,v_2,...,v_{k-1}$的$k$-1条边中随机选择$m$条边，将其中最短的边添加到$T$

在线随机算法处理第$k$个到来的顶点的至多计算$m$次距离，故算法的新数据响应时间为$\Theta(m)$。算法处理规模为$n$的问题实例的至多计算$mn$次距离，故算法的时间复杂度为$\Theta(mn)$。如果$m$是常数，则在线随机算法将在常数时间内完成对每个新到来的数据的处理。如果$m=n-1$,则在线随机算法即为11.2.1中的在线贪心算法。

下面先定义随机在线算法的竞争度，然后分析算法OnlineRandMST($m$)的竞争度。

**定义11.2**.设问题$A$是一个输入数据在线到来的计算问题，算法$\mathcal{A}$是求解问题$A$的一个在线随机算法。对于问题$A$的任意实例$I_A$, 记$E(C_{on}(I_A))$是算法$\mathcal{A}$在实例$I_A$上获得的解的代价$C_{on}(I_A)$的数学期望，$C_{off}(I_A)$表示最优离线算法在实例$I_A$获得的解的代价。如果$E(C_{on}(I_A))\le\alpha\cdot C_{off}(I_A)+\beta$对任意实例$I_A$成立，其中$\beta$是与输入规模$|I_A|$无关的常量，则称在线随机算法$\mathcal{A}$的**竞争度**为$\alpha$。

下面分析求解欧几里德生成树问题的在线随机算法OnlineRandMST($m$)的竞争度。我们将证明OnlineRandMST($m$)的竞争度$\Theta(n)$（定理11.5）。分析过程分为两步。首先，引理11.3将证明欧几里德生成树问题的任意随机算法的竞争度至多为$n$-1，进而算法OnlineRandMST($m$)的竞争度为$O(n)$。然后，引理11.4将证明算法OnlineRandMST($m$)在问题的一个特殊实例上的竞争度为$\Omega(n)$,进而得到算法OnlineRandMST($m$)的竞争度为$\Omega(n)$。

**引理11.3.** 求解欧几里德生成树问题的任意在线算法的竞争度至多为$n$-1。

**证明**.记所有输入点$v_1,v_2,...,v_n$中任意两点间距离的最大值为$d$。

一方面，在最小生成树$T^*$中，输入点集中任意两点是连通的，且距离满足三角不等式，故$T^*$的代价$C_{off}$至少为$d$,即$C_{off}\ge d$。

另一方面，在任意在线算法输出的生成树$T_{on}$中，恰有$n$-1条边且每条边的代价均不超过$d$,故生成树$T_{on}$的代价$C_{on}$至多为$(n-1)d$,亦即$C_{on}\le(n-1)d\le(n-1)C_{off}$。

联立上述两个不等式得，$C_{on}\le(n-1)C_{off}$对任意在线算法成立。 □

**引理11.4.** 算法OnlineRandMST($m$)竞争度为$\Omega(n)$。

**证明.**为了得到引理所述的竞争度下界，考虑算法OnlineRandMST($m$)在图11.2所示的问题实例上的运行。问题实例中在线到来的所有输入点位于区间[-1/2,1/2]内，且如果$i$是奇数，则第$i$个点为$\frac{1}{2}-\frac{i-1}{2}\varepsilon$；如果$i$是偶数，则第$i$个点为$-\frac{1}{2}+\frac{i-2}{2}\varepsilon$，其中$\varepsilon$是一个任意小的正实数。为使得分析清晰，将第$i$个输入点简记为$i$;用$d(i,j)$表示点$i$和点$j$之间的距离。$N(i,k)\in\{1,2,...,i-1\}$表示输入点1,2,...,$i$-1中距离输入点$i$第$k$近的点，其中$1\le k<i$。不难验证

$$N(i,j)=\begin{cases} i-2j & \text{若} 1\le j\le\lfloor(i-1)/2\rfloor \\ i-j+3 & \text{若}\lfloor(i-1)/2\rfloor+1\le j\le i-1\end{cases}$$

且

$$d(i,N(i,j))=\begin{cases} j\cdot\varepsilon\approx 0 & \text{若} 1\le j\le\lfloor(i-1)/2\rfloor \\ 1-\lfloor(i-1)/2\rfloor\cdot\varepsilon+(j-i+1)\cdot\varepsilon\approx 1 & \text{若}\lfloor(i-1)/2\rfloor+1\le j\le i-1\end{cases}$$

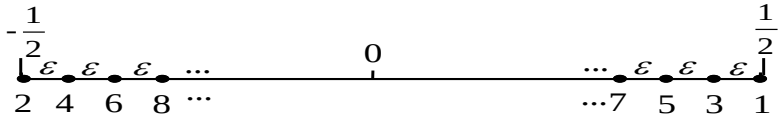



图11.2 证明算法OnlineRandMST($m$)竞争度下界的问题实例

易知，问题实例上最小生成树代价$C_{off}=1-\varepsilon\approx 1$。

下面，分析算法OnlineRandMST($m$)在上述实例上获得的生成树代价$C_{on}$的数学期望$E(C_{on})$。考虑算法对输入点$i$的处理。

当$2\le i\le m+1$时，算法将$i$与1,2,...,$i$-1中距离$i$最近的点（即$N(i,1)$）相连，因此$C_{on}$增加$d(i,N(i,1))$。

当$i>m+1$时，算法将$i$连接到从1,2,...,$i$-1随机选择的$m$个点中距离$i$最近的点，因此$C_{on}$的增量可能是$d(i,N(i,1))$, $d(i,N(i,2))$,..., $d(i,N(i,i-m))$中的任何一个。$C_{on}$的增量为$d(i,N(i,j))$，当且仅当随机选择的$m$个点中距离$i$最近的是$N(i,j)$，而这个随机事件发生的概率为$\binom{i-1-j}{m-1}\Big/\binom{i-1}{m}$，其中分母表示从{1,2,...,$i$-1}选取$m$个点的方案数，分子表示$N(i,1),N(i,2),...,N(i,j-1)$未被选中且$N(i,j)$被选中而其余$m$-1个点从剩余元素任意选取的方案数。

由上面的分析可知，

$$
\begin{aligned}
E(C_{on}) &= \sum_{i=2}^{m+1} d(i,N(i,1)) + \sum_{i=m+2}^{n}\sum_{j=1}^{i-m} d(i,N(i,j))\cdot\binom{i-1-j}{m-1}\Big/\binom{i-1}{m} \\
&\approx 1+\sum_{i=m+2}^{n}\sum_{j=\left\lfloor\frac{i-1}{2}\right\rfloor+1}^{i-m}\binom{i-1-j}{m-1}\Big/\binom{i-1}{m} \\
&= 1+\sum_{i=m+2}^{n}\sum_{j=\left\lfloor\frac{i+1}{2}\right\rfloor}^{i-m}\binom{i-1-j}{m-1}\Big/\binom{i-1}{m} && \text{（因为}\lfloor x+1\rfloor=\lfloor x\rfloor+1\text{）} \\
&= 1+\sum_{i=m+2}^{n}\binom{i-1}{m}^{-1}\sum_{j=\left\lfloor\frac{i+1}{2}\right\rfloor}^{i-m}\binom{i-1-j}{m-1} \\
&= 1+\sum_{i=m+2}^{n}\binom{i-1}{m}^{-1}\sum_{k=0}^{\lfloor(i-1)/2\rfloor-m}\binom{m-1+k}{m-1} && \text{（令}k=i-m-j\text{）} \\
&= 1+\sum_{i=m+2}^{n}\binom{i-1}{m}^{-1}\sum_{k=0}^{\lfloor(i-1)/2\rfloor-m}\binom{m-1+k}{k} && \text{（因为}\binom{m+k}{k}=\binom{m+1+n}{m+1}\text{）} \\
&= 1+\sum_{i=m+2}^{n}\binom{\lfloor i-1\rfloor/2}{m}\Big/\binom{i-1}{m} && \text{（因为}\sum_{k=0}^{n}\binom{m+k}{k}=\binom{m+1+n}{m+1}\text{）} \\
&\ge 1+\sum_{i=2m+1}^{n}\binom{\lfloor i-1\rfloor/2}{m}\Big/\binom{i-1}{m} && \text{（}m\ge 1\text{蕴涵}2m+1\ge m+2\text{）} \\
&= 1+\sum_{i=2m+1}^{n}\frac{\lfloor(i-1)/2\rfloor(\lfloor(i-1)/2\rfloor-1)...(\lfloor(i-1)/2\rfloor-m+1)}{(i-1)(i-2)...(i-m)} \\
&> 1+\sum_{i=2m+1}^{n}\frac{((i-1)/2-1)((i-1)/2-2)...((i-1)/2-m)}{(i-1)(i-2)...(i-m)} && \text{（因为}\lfloor x\rfloor>x-1\text{）} \\
&= 1+\frac{1}{2^m}\sum_{i=2m+1}^{n}\frac{(i-3)(i-5)...(i-2m-1)}{(i-1)(i-2)...(i-m)} \\
&= 1+\frac{1}{2^m}\sum_{i=2m+1}^{n}\left(1-\frac{2}{i-1}\right)\left(1-\frac{3}{i-2}\right)...\left(1-\frac{m+1}{i-m}\right)
\end{aligned}
$$

$$\geq 1+\frac{1}{2^m}\sum_{i=2m+1}^{n}\left(1-\sum_{k=1}^{m}\frac{k+1}{i-k}\right)$$

$$\geq 1+\frac{1}{2^m}\sum_{i=2m+1}^{n}\left(1-\sum_{k=1}^{m}\frac{m+1}{i-k}\right) \qquad \left(\frac{k+1}{i-k}\leq\frac{m+1}{i-k}\right)$$

$$\geq 1+\frac{n-2m}{2^m}-\frac{m+1}{2^m}\sum_{i=2m+1}^{n}\sum_{k=1}^{m}\frac{1}{i-k}$$

$$=1+\frac{n-2m}{2^m}-\frac{m+1}{2^m}\sum_{i=2m+1}^{n}[H(i-1)-H(i-m-1)] \qquad \left(H(j)=\sum_{k=1}^{j}1/j\right)$$

$$=1+\frac{n-2m-o(n)}{2^m} \qquad (m\text{是常数})$$

由此可知，

$$\frac{E(C_{on})}{C_{off}}\geq 1+\frac{n-2m-o(n)}{2^m}$$ □

由引理11.3和引理11.4立刻得到定理11.5.

**定理11.5.** 算法OnlineRandMST($m$)竞争度为$\Theta(n)$。 □

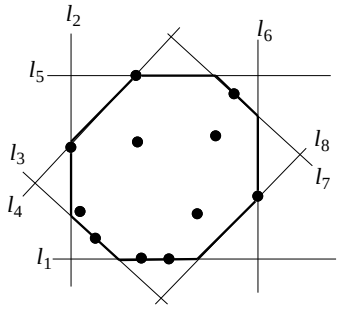
## 11.3 凸包在线算法

凸包问题要求输出包含平面上给定$n$个点$P_1,...,P_n$的最小凸多边形（参见3.6节）。本节讨论在线凸包问题的求解，亦即输入点$P_1,...,P_n$是在线到来的，并且当$P_i$到来时要求输出$P_1,...,P_i$的一个近似凸包。

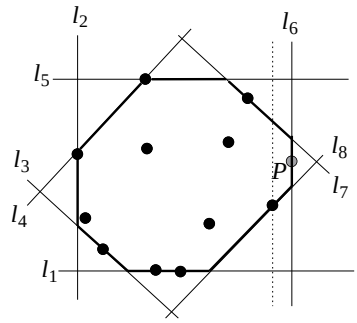
试想一下，如果在线算法维护了$P_1,...,P_{i-1}$的凸包$A$，当$P_i$到来时如何更新凸包$A$使其成为$P_1,...,P_{i-1},P_i$的凸包。如果点$P_i$位于$A$的内部或边界上，则$A$即是$P_1,...,P_{i-1},P_i$的凸包；故维护过程无需任何操作。如果$P_i$位于$A$的外部，则需要调整$A$的部分边界使得$P_i$位于调整后的凸多边形的内部或边界上。困难在于，判断$P_i$是否位于$P_1,...,P_{i-1}$的凸包$A$内和判定调整$A$的哪些边界才能使得调整后的凸包最小，均需要比较复杂的操作过程；这将使得在线算法处理新到来的数据的时间过长。一种自然的想法是，维护$P_1,...,P_{i-1}$的一个近似凸包$A$使得上述两个操作简单可行。

给定参数$m$,利用$m$对斜率分别为0,$\tan(\pi/m)$, $\tan(2\pi/m)$,..., $\tan((m-1)\pi/m)$的平行线可以很方便地近似维护输入点集的凸包。如图11.3(a)所示，四对平行线给出了所有输入点的边界。当一个新的输入点$P$在线到来时，如果$P$位于平行线围成的凸多边形内部或边界，则无需对边界进行维护；否则，通过移动若干对平行线中靠近$P$的一条直线使得平行线围成的凸多边形包含$P$（如图11.3(b)所示）。为了使得平行线围成的边界尽可能紧凑，维护过程需要遵循两个原则。其一，所有输入点必须位于每对平行线之间。其二，每条直线至少经过一个输入点。

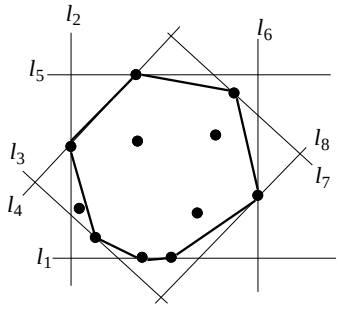
利用平行线围成的边界，可如下输出近似凸包。维护每条直线上距离最远的两个输入点（若直线上仅有一个输入点，则认为距离最远的两个点是同一个点）。将每条直线上维护的两个点依次按逆时针顺序连接即为近似凸包。例如，图11.3(c)给出了图11.3(a)所示边界得到的近似凸包，图11.3(d)给出了图11.3(b)所示边界得到的近似凸包。



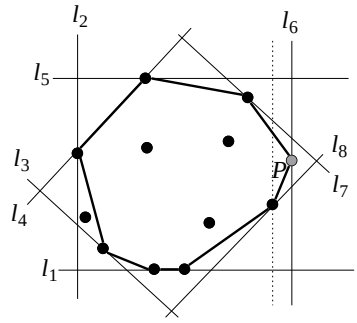

(a)由四组平行线组成的边界




(b)新增输入点之后平移直线




(c)由四组平行线给出的近似凸包




(d)新增输入点之后的近似凸包

图11.3 凸包在线算法示意图

根据上面的思想，可得到如下的凸包在线算法。

---

**凸包在线算法OnlineConHull(*m*)**

**输入**：平面上在线到来的点$P_1,P_2,...,P_n$和近似计算凸包的平行线对数$m$

**输出**： $P_1,P_2,...,P_i$到来后，输出其近似凸包

1. Input($P_1$);

2.初始化$m$对过点$P_1$的平行线,第$j$对平行线$l_j,l_{m+j}$的斜率为$\tan(\frac{j-1}{m}\cdot\pi)$, $1\le j\le m$;

3. $Q_j\leftarrow P_1,Q'_j\leftarrow P_1$,其中$1\le j\le m$; /*$Q_j,Q'_j$记录$l_j$上距离最远的两个输入点*/

4.For $i\leftarrow 2$ To $n$ Do

5. Input($P_i$);

6. For $j\leftarrow 1$ To $m$ Do

7. If $P_i$不在$l_j$和$l_{j+m}$之间 Then 平移$l_j$和$l_{j+m}$中距离$P_i$较近者使其通过$P_i$;

8. ElseIf $P_i\in l_j$（或$l_{m+j}$） Then 更新$Q_j,Q'_j$（或$Q_{m+j},Q'_{m+j}$）;

9. 顺序连接位于直线$l_1,l_2,...,l_{2m}$上保留的（至多两个）点构成$P_1,...,P_i$的近似凸包;

---

算法OnlineConHull第7步和第8步的时间开销为常数（请读者参照3.6节的方法给出实现细节）；第9步的时间开销为$O(m)$,因为每条直线上仅维护两个相距最远的输入点。因此，算法OnlineConHull对新到达的每个数据的响应时间为$O(m)$；算法的总时间复杂度为$O(mn)$。

用多边形的周长表示其代价，下面分析算法OnlineConHull的竞争度。分析过程涉及三个多边形，如图11.4(a)所示。

**多边形*E***：由$m$对平行线围成的多边形$\langle A_1,A_2,...,A_{2m}\rangle$,其中$A_i$是$l_{i-1}$和$l_i$的交点

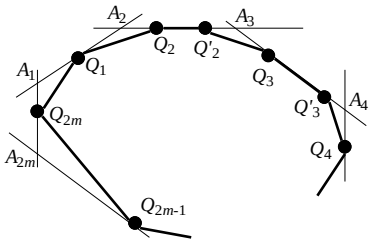
**多边形*A***：近似解多边形$\langle Q_1,Q'_1,,...,Q_{2m}, Q'_{2m}\rangle$, 其中$Q_i,Q'_i$是$l_i$上距离最远的输入点

**多边形*C*:** 输入点$P_1,...,P_n$的凸包

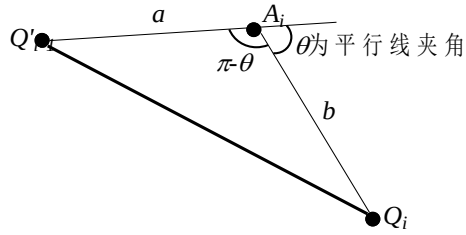
将多边形$E$,$A$,$C$的周长分别记为$L(E)$,$L(A)$和$L(C)$。根据算法的操作规则知，多边形$C$介于多边形$E$和多边形$A$之间（参见图11.3）。因此，$L(E) \geq L(C) \geq L(A)$。于是，由

$$\frac{L(C)}{L(A)} \leq \frac{L(E)}{L(A)}$$

可知，$L(E)/L(A)$给出竞争度的一个上界。




(a)平行线围成的多边形与近似解的关系




(b)平行线与近似解之间的三角形

图11.4 凸包在线算法竞争度分析示意图

为了比较多边形$E$和多边形$A$的周长，考虑两个多边形之间的三角形$\Delta A_i Q_i Q'_{i-1}$，如图11.4所示。值得强调的是，如果直线$l_i$上仅有一个输入点，则$Q_i = Q'_{i-1}$。由三角形的余弦定理可得，

$$\left(\frac{|Q'_{i-1}A_i| + |A_iQ_i|}{|Q'_{i-1}Q_i|}\right)^2 = \frac{a^2+b^2+2ab}{a^2+b^2+2ab\cos\theta}$$

$$= \frac{1+2ab/(a^2+b^2)}{1+\cos\theta \cdot 2ab/(a^2+b^2)}$$

$$\leq \frac{1+1}{1+\cos\theta\cdot 1} \qquad \left(\frac{1+t}{1+\cos\theta\cdot t}\text{在}[0,1]\text{上是增函数}\right)$$

$$= \sec^2\frac{\theta}{2}$$

因此，$|Q'_{i-1}A_i|+|A_iQ_i| \leq \sec\frac{\theta}{2}\cdot|Q'_{i-1}Q_i|$。进而得到，

$$\begin{aligned} L(E) &= |A_1A_2|+|A_2A_3|+\ldots+|A_{2m}A_1| \\ &= |A_1Q_1|+|Q_1Q'_1|+|Q'_1A_2|+|A_2Q_2|+|Q_2Q'_2|+|Q'_2A_3|+\ldots+|A_{2m}Q_{2m}|+|Q_{2m}Q'_{2m}|+|Q'_{2m}A_1| \\ &\leq \sec\frac{\theta}{2}\cdot(|Q_1Q'_1|+|Q'_1Q_2|+|Q_2Q'_2|+|Q'_2Q_3|+\ldots+|Q_{2m}Q'_{2m}|+|Q'_{2m}Q_1|) \\ &= \sec\frac{\theta}{2}\cdot L(A) \end{aligned}$$

亦即，$L(E)/L(A) \leq \sec\frac{\theta}{2}$。

综上所述，再注意到$\theta=\pi/m$,得到下面的定理。

**定理11.6.** 算法OnlineConHull的竞争度为 $\sec\frac{\pi}{2m}$。 □

## 11.4 线性链表在线更新算法

给定含$n$个数据的线性链表$L$,访问链表元素$x$时需要从链表头部开始顺序扫描直到到达元素$x$,因此访问元素$x$的代价等于$x$在链表中的位置$rank(L,x)$。于是，如果$L$中任意元素$x$被访问的概率为$p(x)$,则$L$被访问的平均代价 $cost_{avg}(L)$为

$$Cost_{avg}(L)=\sum_{x\in L} p(x)\cdot rank(L,x)$$

平均代价 $cost_{avg}$反映了链表上一系列访问操作的总代价。不难发现，为使得$Cost_{avg}(L)$达到最小值，访问概率$p(x)$越大的元素$x$在$L$中的位置应该越靠前,即$rank(L,x)$越小；访问概率$p(x)$越小的元素$x$在$L$中的位置应该越靠后,即$rank(L,x)$越大。

如果链表$L$的访问操作序列已知，则可以统计每个元素$x$被访问的频率，然后按频率递减的顺序将所有元素依次存储在链表中。将这种方式组织的链表记为$L^*$,它可以确保在该访问操作系列上的总代价达到最小值。

然而，在实际应用中情况却是：数据被组织成一个链表$L$，且访问操作序列以在线方式逐渐到来，并允许访问链表$L$中任意元素之后重新组织链表$L$。如何才能使得访问链表$L$的总代价尽可能的小？即，需要利用在线算法求解如下计算问题。

**线性链表在线更新问题**

**输入**：存储$n$个数据的线性链表$L$和在线到达的$m$个访问请求

**输出**：以在线方式维护的线性链表$L$使得平均访问代价尽可能低

线性链表在线更新问题可以直观地以如下方式在线求解。初始化链表$L$时将所有元素的访问频率置为0；每个在线到达的访问请求（不妨设访问元素$x$）被响应之后，将$x$的访问频率增加1并调整$x$在$L$中的存储位置使得$L$中所有元素按访问频率递减有序；维护链表$L$时依次交换$x$和其前驱元素直到$x$的频率小于等于前驱元素的频率。这种在线处理策略的一个明显缺陷是，在最坏情况下响应一次访问请求需要$2n$次操作。例如，请求访问链表的最后一个元素时，顺序扫描到达链表最后一个元素$x$的需要$n$个操作，$x$的频率增加后可能变成链表中频率最大的元素，因此维护链表需要$n$个操作。能否改进这种策略以降低响应单次访问请求的最坏时间复杂度呢？

上述在线更新算法可以如下改进。由于无法预知对$L$将来的访问，每次访问链表中元素$x$时，“理所当然”地认为$x$将成为访问频率最高的元素，将$x$前置使其成为$L$的第一个元素。于是，得到如下的线性链表在线访问算法。

---

**线性链表在线访问算法OnlineAccess($L$,$x$)**

**输入**：链表$L$和在线到达的访问请求$x$　/*$L$中元素以任意顺序被初始化*/

**输出**：访问$x$并输出更新后的链表

1.顺序扫描$L$,访问$x$；

2.将$x$的前驱的后继指针指向$x$的后继，然后将$x$置为链表的一个位置；

---

显然，算法OnlineAccess($L$,$x$)响应对$x$的访问请求的开销取决于$x$在链表中的位置$rank(L,x)$+1，因为算法第2步的开销是常数。

**定理11.7**.　算法OnlineAccess的竞争度为2。

**证明**：为了得到算法OnlineAccess的竞争度，将算法OnlineAccess维护的链表$L$与所有访问请求已知时利用离线算法维护的最优线性链表$L^*$进行比较，其中$L^*$的精确定义如前所述。注意，$L$和$L^*$存储的元素相同，区别仅在于每个元素在链表中的存储位置。因此，$L$中元素顺序与$L^*$中元素顺序的差别体现了在线算法和离线算法的差别。我们利用元素顺序在$L$和$L^*$中的差别定义$L$的势能函数（参见第6章），通过势能分析来获得算法OnlineAccess的竞争度。

为此，将算法OnlineAccess维护的初始链表记为$L_0$,响应第$i$次在线访问请求之后的链表记为$L_i$。定义$L_i$的势能$\Phi(L_i)$为

$$\Phi(L_i)=L_i\text{相对于}L^*\text{的反序对个数}$$
$$=|\{(y,z)|\ y<_{L_i} z\text{但}z<_{L^*} y\}|$$

其中，$y<_{L_i} z$表示在$L_i$中元素$y$位于元素$z$之前，$z<_{L^*} y$表示在$L^*$中$z$位于$y$之前。例如，对于如下的线性链表$L^*$和$L_i$，$\Phi(L_i)=|\{(35,10),(35,15),(35,20),(35,30),(30,20)\}|=5$。


设第$i$个在线到达的访问请求访问元素$x$,它将线性链表$L_{i-1}$更新成$L_i$。正如前面分析结果所述，此次操作的实际代价为

$$c_i=rank(L_{i-1},x)+1$$

为了分析此次操作的平摊代价$\alpha_i$，如图11.5所示，将链表中除$x$之外的所有元素划分为如下四个集合

$$A=\{y|\ y<_{L_{i-1}} x\text{但}y<_{L^*} x\},$$
$$B=\{y|\ y<_{L_{i-1}} x\text{但}x<_{L^*} y\},$$
$$C=\{y|\ x<_{L_{i-1}} y\text{但}y<_{L^*} x\},$$
$$D=\{y|\ x<_{L_{i-1}} y\text{但}x<_{L^*} y\}$$


图11.5　线性链表$L_{i-1}$,$L_i$和$L^*$中元素位置关系分类示意图

于是，$\Phi(L_{i-1})=|B|+|C|$,且$\Phi(L_i)=|A|+|C|$。进而，$\Phi(L_i)-\Phi(L_{i-1})=|A|-|B|$。因此

$$\begin{aligned}\alpha_i &= c_i+\Phi(L_i)-\Phi(L_{i-1})\\ &= rank(L_{i-1},x)+1+(|A|-|B|)\\ &= (|A|+|B|+1)+1+(|A|-|B|) && (rank(L_{i-1},x)=|A|+|B|+1)\\ &=2(|A|+1)\\ &\le 2\cdot rank(L^*,x) && (|A|+1\le rank(L^*,x))\end{aligned}$$

因此，在含 $n$ 个元素的线性链表 $L_0$ 上连续执行 $m$ 个访问操作的总平摊代价为

$$\begin{aligned}\sum_{i=1}^{m}\alpha_i &= \sum_{i=1}^{m}[c_i+\Phi(L_i)-\Phi(L_{i-1})] && (\text{设第 } i \text{ 次操作访问 } x_i)\\ &=\sum_{i=1}^{m}c_i+[\Phi(L_m)-\Phi(L_0)]\\ &\le 2\sum_{i=1}^{m}rank(L^*,x_i)\end{aligned}$$

其中，$\sum_{i=1}^{m}rank(L^*,x_i)$ 是在最优链表 $L^*$上执行相同操作序列的总代价。再由势能函数 $\Phi$ 的定义可知，$0\le\Phi(L_i)\le n(n-1)/2$ 对任意 $L_i$ 成立，故$|\Phi(L_m)-\Phi(L_0)|=O(n^2)$，其中 $n$ 相对于操作序列的长度 $m$ 而言是固定值。于是，$m$ 个在线访问操作的总代价满足

$$\begin{aligned}\sum_{i=m}^{n}c_i &\le 2\sum_{i=m}^{n}rank(L^*,x_i)+|\Phi(L_m)-\Phi(L_0)|\\ &\le 2\sum_{i=m}^{n}rank(L^*,x_i)+O(n^2)\end{aligned}$$

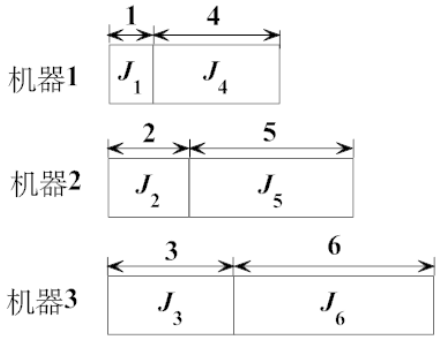
□

## 11.5 最短并行调度在线算法

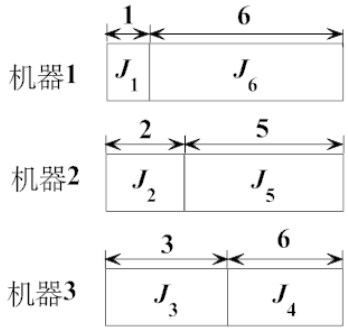
最短并行调度问题要求将执行时间分别为 $a_1,...,a_n$ 的 $n$ 个计算任务在 $m$ 台完全一

样的计算机上进行调度使得并行时间最短。最短并行在线调度问题中，$n$ 个任务是在线到来的，并且要求每个任务到来后即刻完成其调度。

10.2.3 给出了求解最短并行调度问题的一个 2-近似算法 MinMakespanScheduling，它实际上也是基于贪心思想的一个在线调度算法。该算法贪心地将每个在线到来的任务分配给运行时间最短的机器。例如，对于 3 台机器和在线到来的长度分别为 1,2,3,4,5,6 的 6 个任务 $J_1,J_2,J_3,J_4,J_5,J_6$，该算法的调度结果如图 11.6(a)所示,并行调度的总长度为 9。采用 10.2.3 节类似的分析，可以证明该算法的竞争度为 2-1/$m$。




(a)贪心调度的结果




(b)适中调度的结果

**图 11.6** 两种策略在 3 台机器和长度分别为 1,2,3,4,5,6 的任务 $J_1,..,J_6$ 上的调度结果

如果不采用贪心策略，而改用适中策略，亦即将每个新到来的任务分配给执行时间长度适中（既不是最长的也不是最短）的机器，而不是将每个新到来的任务始终分配给执行时间长度最短的机器，则有望改进在线算法的竞争度。将改进后的算法称为适中在线调度算法 OnMinSpan。例如，对于图 11.6(a)所示的 6 个任务，改用 OnMinSpan 算法进行在线调度，6 个任务的并行时间长度将达到 7，如图 11.6(b)所示。

在如下的适中在线调度算法中，取 $\varepsilon$=1/70 是常数，取 $\in$[0.445-1/2$m$,0.445+1/2$m$] 使得 $\delta m$ 是整数。调度过程中，每台机器所执行的所有任务的时间长度总和称为该机器的高度，所有机器的高度初始化为 0。算法利用两个数组 $F$[1:$\delta m$]和 $L$[1:$m$-$\delta m$]维护 $m$ 台机器及其高度，并且调度过程确保任意时刻 $F$[1],$F$[2],...,:$F$[$\delta m$],$L$[1],...,$L$[$m$-$\delta m$]中维护的所有高度顺序递增（第 3 步）。当第 $i$ 个任务到来后，检查 $F$[1:$\delta m$]中所有高度的平均值 $A(F)$和 $L$[1:$m$-$\delta m$]中维护的第 1 个高度 $L$[1]是否满足条件：$L[1]+a_i \le (2-\varepsilon)\cdot A(F)$，其中 $a_i$ 是第 $i$ 个任务的执行时间长度。若 $L[1]+a_i \le (2-\varepsilon)\cdot A(F)$，则将第 $i$ 个任务分配给 $L$[1:$m$-$\delta m$]中高度最小的机器（亦即 $L$[1]维护的机器），然后令 $L[1]=L[1]+a_i$ 并维护 $L$[1:$m$-$\delta m$]成递增顺序（第 6 步）。若 $L[1]+a_i>(2-\varepsilon)\cdot A(F)$，则将第 $i$ 个任务分配给 $F$[1:$\delta m$]中高度最小的机器（亦即 $F$[1]对应的机器），然后令 $F[1]=F[1]+a_i$ 并维护 $F$[1:$\delta m$]和 $L$[1:$m$-$\delta m$]使得 $F$[1],$F$[2],...,:$F$[$\delta m$],$L$[1],...,$L$[$m$-$\delta m$]顺序递增（第 7 步）。

注意，实现算法时数组 $F$[1:$\delta m$]和 $L$[1:$m$-$\delta m$]可以使用一个数组，以便于数组的维护。算法描述时使用两个数组的目的是为了与分析过程中使用的记号一致，以便于对分析过程的理解。

**适中在线调度算法 OnMinSpan(*m*)**

**输入**：$m$ 台相同机器和在线到来的 $n$ 个执行时间分别为 $a_1,a_2,...,a_n$ 的任务，$m\ge 70$

**输出**：$n$ 个任务在 $m$ 台机器上的一个并行调度，使得并行时间尽可能短

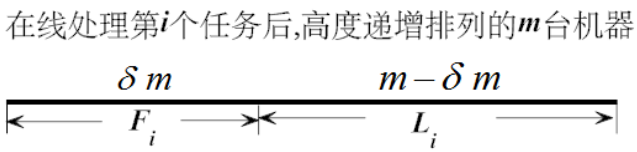
1.$\varepsilon \leftarrow 1/70$;

2.取 $\delta \in$[0.445-1/2$m$,0.445+1/2$m$]使得 $\delta m$ 是整数；

3.$F[1{:}\delta m]$维护 $\delta m$ 台机器的高度，$L[1{:}m{-}\delta m]$维护 $m{-}\delta m$ 台机器的高度,高度初始化为 0；
/*$F[1],F[2],...,:F[\delta m],L[1],...,L[m{-}\delta m]$中维护的所有高度顺序递增*/
4. For $i\leftarrow 1$ To $n$
5. Input($a_i$);
6. If $L[1]+a_i \le (2-\varepsilon)\cdot A(F)$ Then /*$A(F)$ 表示 $F[1{:}\delta m]$中所有高度的平均值*/
将 $a_i$ 分配给 $L[1]$对应的机器并维护 $L[1{:}m{-}\delta m]$为高度递增顺序；
7. Else 将 $a_i$ 分配给 $F[1]$对应的机器并维护 $F[1{:}\delta m],L[1{:}m{-}\delta m]$为高度递增顺序；

显然，算法OnMinSpan($m$)响应第$i$个任务的调度请求的时间开销为$O(m)$。因此，算法的时间复杂度为$O(mn)$。

下面证明，算法OnMinSpan($m$)的竞争度为2-ε≈1.986;由此可见，当$m$≥70时算法OnMinSpan($m$)优于10.2.3中的MinMakespanScheduling。竞争度的分析过程很难很长，分析过程的思路是，假设算法的竞争度大于2-ε，则必然存在一个问题实例$I=\langle a_1,a_2,...,a_{t+1}\rangle$使得算法OnMinSpan($m$)在实例$I$上产生的近似解的代价$On_{t+1}$大于实例$I$上的优化解代价$OPT_{t+1}$的2-ε倍。根据自然数的良序公理，如果这种实例确实存在，则必然存在一个长度最短的反例。分析过程通过研究算法OnMinSpan的最短(2-$\varepsilon$)-竞争反例$I=\langle a_1,a_2,...,a_{t+1}\rangle$的性质（引理11.8-11.10），最后证明这样的反例不存在，进而得到算法的竞争比小于2-ε（定理11.11）；算法实际运行过程的竞争比通常小于2-ε，例如图11.6(b)的输出是该实例的最优解。分析过程中使用的记号及其含义如图11.7所示。




$M_i$ —$m$ 台机器的最小高度　　$A_i$—$m$ 台机器的平均高度
$On_i$—在线算法并行时间长度　　$OPT_i$—最优解并行时间长度，$OPT_i \ge A_i$
$A(F_i)$—$F_i$ 中高度的平均值,$A(F_i)\le A_i$　　$A(L_i)$—$L_i$ 中高度的平均值, $A(L_i)\ge A_i$
$M(F_i)$—$F_i$ 中高度的最小值，　$M(F_i)=M_i$　　$M(L_i)$—$L_i$ 中高度的最小值, $M(L_i)\ge M_i$

图 11.7 算法 OnMinSpan($m$)竞争度分析过程采用的符号和含义示意图

**引理 11.8.** 在算法 OnMinSpan 的最短(2-$\varepsilon$)-竞争反例 $I=\langle a_1,a_2,...,a_{t+1}\rangle$ 上，$On_{t+1}=M_t+a_{t+1}$。

**证明**: 考虑第 $t$+1 个任务到来时。如果 $M(L_t)+a_{t+1}\le(2-\varepsilon)\cdot A(F_t)$, 则

$$\begin{aligned} On_{t+1} &= M(L_t)+a_{t+1} \\ &\le (2-\varepsilon)\cdot A(F_t) \\ &\le (2-\varepsilon)\cdot A_t \\ &\le (2-\varepsilon)\cdot A_{t+1} \\ &\le (2-\varepsilon)\cdot OPT_{t+1} \end{aligned}$$

这与 $I=\langle a_1,a_2,...,a_{t+1}\rangle$ 是算法的(2-$\varepsilon$)-竞争反例矛盾。

因此，算法必然将 $t$+1 个任务分配给高度最小的机器，即 $On_{t+1}=M_t+a_{t+1}$。 □

**引理 11.9：** 在算法 OnMinSpan 的最短(2-$\varepsilon$)-竞争反例 $I=\langle a_1,a_2,...,a_{t+1}\rangle$ 上，$M_t>(1-\varepsilon)A_t$。

**证明**. 如果 $M_t\le(1-\varepsilon)A_t$, 由引理 11.8 可得，

$$\begin{aligned} On_{t+1} &= M_t+a_{t+1} \\ &\le (1-\varepsilon)A_t+a_{t+1} \\ &\le (1-\varepsilon)OPT_{t+1}+OPT_{t+1} \qquad (OPT_{t+1}\ge A_{t+1}\ge A_t \text{ 且 } OPT_{t+1}\ge a_{t+1}) \\ &= (2-\varepsilon)OPT_{t+1} \end{aligned}$$

这与 $I=\langle a_1,a_2,...,a_{t+1}\rangle$ 是算法的(2-$\varepsilon$)-竞争反例矛盾。 □

**引理 11.10.**设 $I=\langle a_1,a_2,...,a_{t+1}\rangle$ 是算法的最短 2-竞争反例，则算法 OnMinSpan 处理第 $t$ 个任务之后，每台机器至少处理一个长度至少为 $\frac{1}{2(1-\varepsilon)}A_t \geq \frac{1}{2(1-\varepsilon)}M_t$ 的任务。

**证明**.基本思想是利用引理 11.9 所述性质，将机器分类之后证明每一类机器均满足引理 11.10，从而完成引理的证明。为了分类机器，引入常数 $\tau\in[0.14-1/(2\delta m),0.14+1/(2\delta m)]$ 并定义在线算法 OnMinSpan 在实例 $I$ 上操作的 3 个时刻（如图 11.8 所示）。第一个时刻记为 $r$ 时刻，算法完成第 $r$ 个任务的调度之后 $F_r$ 中所有机器的高度均小于等于(1-$\varepsilon$)$A_t$ 而 $L_r$ 所有机器的高度均大于(1-$\varepsilon$)$A_t$；第二个时刻记为 $s$ 时刻,算法完成第 $s$ 个任务的调度之后高度最小的 $\lfloor\tau\delta m\rfloor$ 台机器的高度均小于等于(1-$\varepsilon$)$A_t$ 且其余机器的高度均大于(1-$\varepsilon$)$A_t$，并令集合 $R_s$ 记录 $s$ 时刻高度最小的$\lfloor\tau\delta m\rfloor$ 台机器；第三个时刻是第 $t$ 个任务被在线算法处理完的时刻，即 $t$ 时刻，由引理 11.9 可知，所有机器的高度在 $t$ 时刻均大于(1-$\varepsilon$)$A_t$。

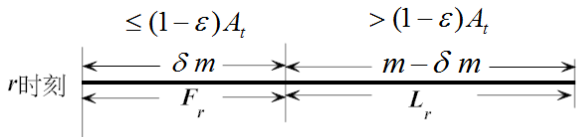
由算法 OnMinSpan 的操作过程可知，初始时所有机器的高度均为 0，此后每台机器的高度随时间推移单调递增，且每个新任务到来后只有一台机器的高度增大。因此，$r$ 时刻和 $s$ 时刻存在且 $r<s<t$。

下面，证明如下三个论断，进而完成引理 11.10 的证明。

**论断 1**. 在 $t$ 时刻，$R_s$ 中每台机器至少执行一个长度至少为 $\frac{1}{2(1-\varepsilon)}A_t$ 的任务。

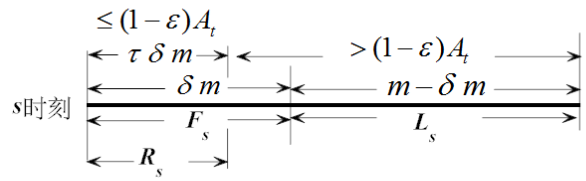
**论断 2**. 在 $t$ 时刻，$F_r$-$R_s$ 中每台机器至少执行一个长度至少为 $\frac{1}{2(1-\varepsilon)}A_t$ 的任务。

**论断 3**. 在 $t$ 时刻，$L_r$ 中每台机器至少执行一个长度至少为 $\frac{1}{2(1-\varepsilon)}A_t$ 的任务。



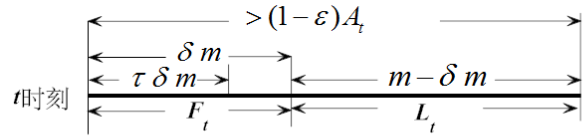

$F_r$所有机器的高度满足$\leq(1-\varepsilon)A_t$

$L_r$所有机器的高度满足$>(1-\varepsilon)A_t$




最小的$\tau\delta m$个高度满足$<(1-\varepsilon)A_t$

$Rs$记录此时刻高度最低的$\tau\delta m$台机器




所有机器的最小高度$M_t$满足 $M_t>(1-\varepsilon)A_t$

$Rs$记录的机器在此时刻可能分布于列表的任意位置

图 11.8　引理 11.10 证明过程中的记号和含义示意图

**论断 1 的证明**。令 $p$ 是 $R_s$ 中任意一台机器。由于 $s$ 时刻 $p$ 的高度小于等于$(1-\varepsilon)A_t$但在 $t$ 时刻 $p$ 的高度大于$(1-\varepsilon)A_t$，因此从 $s$ 时刻到 $t$ 时刻期间 $p$ 至少接受一个任务，记在此期间 $p$ 接受的第一个任务为在线到来的第 $i$+1 个任务，其长度为 $a_{i+1}$，其中 s<$i$<$t$。因此只需证明 $a_{i+1}>\frac{1}{2(1-\varepsilon)}A_t$即可完成论断 1 的证明。

由算法的操作过程知，第 $i$+1 个任务要么分配给 $L_i$ 中高度最小的机器，要么分配给所有机器中高度最小的机器。由于 $p\in R_s$，且在 $i$ 时刻之前 $p$ 未接受过新任务，故 $p$ 不在 $L_i$ 中。因此，在 $i$ 时刻，$p$ 是所有机器中高度最小的机器，且 $M(L_i)+a_{i+1}>(2-\varepsilon)A(F_i)$。由此可得，

$$a_{i+1}>(2-\varepsilon)A(F_i)-M(L_i) \tag{1}$$

此外，由

$$\begin{aligned} A_t &\geq A_i \\ &=\delta\cdot A(F_i)+(1-\delta)\cdot A(L_i) \\ &\geq\delta\cdot A(F_i)+(1-\delta)\cdot M(L_i) \end{aligned}$$

可得

$$M(L_i)\leq\frac{A_t-\delta\cdot A(F_i)}{1-\delta} \tag{2}$$

再由 $F_s$-$R_s$ 中每台机器的高度至少为$(1-\varepsilon)A_t$，得到

$$\begin{aligned} A(F_i)&\geq A(F_s) \\ &=\tau\cdot R_s \text{ 中机器的平均高度}+(1-\tau)\cdot F_s\text{-}R_s \text{ 中机器的平均高度} \\ &\geq\tau\cdot A(R_s)+(1-\tau)\cdot[(1-\varepsilon)A_t] \end{aligned} \tag{3}$$

综上所述，有

$$\begin{aligned} a_{i+1}&>(2-\varepsilon)A(F_i)-M(L_i) && \text{（(1)式）} \\ &>(2-\varepsilon)A(F_i)-\frac{A_t-\delta\cdot A(F_i)}{1-\delta} && \text{（(2)式）} \\ &=(2-\varepsilon+\frac{\delta}{1-\delta})A(F_i)-\frac{1}{1-\delta}A_t \\ &\geq(2-\varepsilon+\frac{\delta}{1-\delta})(\tau\cdot A(R_s)+(1-\tau)\cdot[(1-\varepsilon)A_t])-\frac{1}{1-\delta}A_t && \text{（(3)式）} \\ &=[(2-\varepsilon+\frac{\delta}{1-\delta})(1-\varepsilon)(1-\tau)-\frac{1}{1-\delta}]\cdot A_t+(2-\varepsilon+\frac{\delta}{1-\delta})\tau\cdot A(R_s) \\ &>[(2-\varepsilon+\frac{\delta}{1-\delta})(1-\varepsilon)(1-\tau)-\frac{1}{1-\delta}]\cdot A_t \\ &>\frac{1}{2(1-\varepsilon)}A_t && \text{（}\tau\approx0.14,\ \delta\approx0.445,\varepsilon=1/70\text{）} \end{aligned}$$

**论断 2 的证明**。令 $p$ 是 $F_r$-$R_s$ 中任意一台机器。由于在 $r$ 时刻 $p$ 的高度小于等于$(1-\varepsilon)A_t$但在 $s$ 时刻 $p$ 的高度大于$(1-\varepsilon)A_t$，因此从 $r$ 时刻到 $s$ 时刻期间必存在 $i$ 时刻使得机器 $p$ 接受第 $i$+1 个任务之后其高度首次大于$(1-\varepsilon)A_t$。即，$p$ 在 $i$ 时刻的高度 $h_i(p)\leq(1-\varepsilon)A_t$，$p$

接受第 $i$+1 个任务（长度为 $a_{i+1}$），且 $h_{i+1}(p)=h_i(p)+a_{i+1}>(1-\varepsilon)A_t$。只需证明 $a_{i+1}>\frac{1}{2(1-\varepsilon)}A_t$ 即可，证明的思路是考察高度差 $h_{i+1}(p)-h_i(p)$的下界。

为了在证明过程中方便地使用证明论断 1 时出现的表达式，令

$$\beta_1=(2-\varepsilon+\frac{\delta}{1-\delta})(1-\varepsilon)(1-\tau)-\frac{1}{1-\delta}$$

$$\beta_2=(2-\varepsilon+\frac{\delta}{1-\delta})\tau$$

$$\alpha=\frac{1-\beta_1}{1-\beta_2}+\varepsilon\frac{1-\tau\delta}{\tau\delta(1+\beta_2)}$$

利用 $\tau\approx0.14$, $\delta\approx0.445$,$\varepsilon=1/70$ 容易验证，$\beta_1\geq\frac{\delta}{2(1-\varepsilon)}$ 且 $\alpha\leq1-\varepsilon-\frac{1}{2(1-\varepsilon)}$。

首先，用 $A_t$ 给出 $A(R_s)$的一个界限。由论断 1 的证明过程知，从 $s$ 时刻到 $t$ 时刻，$R_s$ 中每台机器的高度至少增长$\beta_1A_t+\beta_2A(R_s)$，而 $R_s$ 之外的每台机器的高度在 $s$ 时刻至少为$(1-\varepsilon)A_t$ 且此后单调递增。故 $A_t\geq(1-\tau\delta)\cdot(1-\varepsilon)A_t+\tau\delta\cdot[\beta_1A_t+\beta_2A(R_s)]$,由此得到

$$\begin{aligned}A(R_s)&\leq\frac{1-(1-\tau\delta)(1-\varepsilon)-\tau\delta\beta_1}{\tau\delta(1+\beta_2)}A_t\\&=\left[\frac{1-\beta_1}{1-\beta_2}+\varepsilon\frac{1-\tau\delta}{\tau\delta(1+\beta_2)}\right]A_t\\&=\alpha A_t\end{aligned}$$

接下来证明，在接受第 $i$+1 个任务之前机器 $p$ 的高度 $h_i(p)$不超过 $\alpha A_t$,即 $h_i(p)\leq\alpha A_t$。（这也意味着 $F_r$-$R_s$ 中每台机器的初始高度不超过 $\alpha A_t$。）若不然，$h_i(p)>\alpha A_t$。由 $i$ 时刻的定义知 $h_i(p)\leq(1-\varepsilon)A_t$，故 $p$ 必然位于 $F_i$ 中。又由于 $M_i\leq M_s\leq A(R_s)\leq\alpha A_t$，故 $h_i(p)$不可能是 $F_i$ 中高度最低的机器。这与机器 $p$ 在 $i$ 时刻接受第 $i$+1 个任务矛盾。

综上所述，有

$$\begin{aligned}a_{i+1}&=h_{i+1}(p)-h_i(p)\\&>(1-\varepsilon)A_t-h_i(p)\\&\geq(1-\varepsilon)A_t-\alpha A_t\\&=(1-\varepsilon-\alpha)A_t\\&\geq\frac{1}{2(1-\varepsilon)}A_t\end{aligned}$$

**论断 3 的证明**。令 $p$ 是 $L_r$ 中任意一台机器。由于算法初始化时机器 $p$ 的高度为 0，$r$ 时刻机器 $p$ 的高度大于$(1-\varepsilon)A_t$,因此在 $r$ 时刻之前 $p$ 至少接受一个任务。设 $r$ 时刻之前 $p$ 接受的最后一个任务是第 $i$+1 个任务（其长度为 $a_{i+1}$），其中 $0\leq i<r$，只需证明 $a_{i+1}\geq\frac{1}{2(1-\varepsilon)}A_t$即可。证明思路是，先获得机器 $p$ 在 $r$ 时刻的高度 $h_r(p)$的另一个下界，然后利用该下界给出 $a_{i+1}$ 的下界。

在讨论机器 $p$ 在 $r$ 时刻的的高度之前，先给出 $A(F_r)$的一个上界。证明论断 2 时曾得到 $A(R_s)\leq\alpha A_t$，由于 $R_s$ 中机器的高度是单调递增的，因此 $r$ 时刻 $R_s$ 中机器的平均高度也不超过 $\alpha A_t$。证明论断 2 时还曾得到，$F_r$-$R_s$ 中每台机器的初始高度不超过 $\alpha A_t$，因此在 $r$ 时刻的平均高度也不超过 $\alpha A_t$，利用上述两个结论可以得到 $A(F_r)\leq\alpha A_t$。事实

上

$A(F_r)$ = $r$ 时刻 $R_s$ 中机器的平均高度×$\tau$ + $r$ 时刻 $F_r$-$R_s$ 中机器的平均高度×(1-$\tau$)

≤ $\alpha A_t \times \tau + \alpha A_t \times (1-\tau)$

= $\alpha A_t$

现在，利用 $A(F_r) \le \alpha A_t$ 证明 $p$ 在 $r$ 时刻的的高度 $h_r(p) > (2-\varepsilon)A(F_r)$。由于 $p \in L_r$，故

$h_r(p) > (1-\varepsilon)A_t$

$\ge (1-\varepsilon) \cdot \frac{1}{\alpha} A(F_r)$ （$A(F_r) \le \alpha A_t$）

$\ge (2-\varepsilon)A(F_r)$ （$\alpha \le 1-\varepsilon-\frac{1}{2(1-\varepsilon)}$）

接下来，证明 $r$ 时刻之前 $p$ 接受的最后一个（即第 $i$+1 个）任务时 $p \in F_i$。否则，$p \in L_i$，则 $p$ 此时必然是 $L_i$ 中高度最小的机器（即 $h_i(p)=M(L_i)$）且 $M(L_i)+a_{i+1} \le (2-\varepsilon)A(F_i)$。由此得到

$$h_i(p) \le (2-\varepsilon)A(F_i) \le (2-\varepsilon)A(F_r)$$

这与前面得到的 $h_r(p) > (2-\varepsilon)A(F_r)$ 矛盾。

最后，证明 $a_{i+1} \ge \frac{1}{2(1-\varepsilon)} A_t$。由 $p \in F_i$ 且机器 $p$ 接受第 $i$+1 个任务可知，$p$ 是此时所有机器中高度最小的机器（$h_i(p)=M_i$），且 $M(L_i)+a_{i+1} > (2-\varepsilon)A(F_i)$,

$a_{i+1}=h_{i+1}(p)-h_i(p)$

$=h_{i+1}(p)-M_i$

$=h_r(p)-M_i$ （$r$ 时刻前 $p$ 最后接受的任务是第 $i$+1 个任务）

$\ge h_r(p)-M_r$ （$M_i$ 随时间推移单调递增）

$> (1-\varepsilon)A_t - A(F_r)$ （$h_r(p) > (1-\varepsilon)A_t$ 且 $M_r \le A(F_r)$）

$\ge (1-\varepsilon)A_t - \alpha A_t$ （$A(F_r) \le \alpha A_t$）

$=(1-\varepsilon-\alpha)A_t$

$\ge \frac{1}{2(1-\varepsilon)} A_t$ □

**定理 11.11.**算法 OnMinSpan 的竞争度为 2-$\varepsilon$。

**证明**.只需证明算法 OnMinSpan 不存在(2-$\varepsilon$)-竞争反例。若不然，设 $I=\langle a_1,a_2,...,a_{t+1}\rangle$ 是算法的最短(2-$\varepsilon$)-竞争反例。下面，利用引理 11.8-11.10 证明 $a_{t+1} < (1-\varepsilon)M_t$ 和 $a_{t+1} \ge (1-\varepsilon)M_t$ 均不成立，产生矛盾，这意味着算法 OnMinSpan 不存在(2-$\varepsilon$)-竞争反例。

如果 $a_{t+1} < (1-\varepsilon)M_t$, 则

$On_{t+1}=M_t+a_{t+1}$ （引理 11.8）

$<M_t+(1-\varepsilon)M_t$

$=(2-\varepsilon)M_t$

$\le (2-\varepsilon)M_{t+1}$

$\le (2-\varepsilon)OPT_{t+1}$ （$OPT_{t+1} \ge M_{t+1}$）

这与 $I=\langle a_1,a_2,...,a_{t+1}\rangle$ 是算法 OnMinSpan 的(2-$\varepsilon$)-竞争反例矛盾。

如果 $a_{t+1} \ge (1-\varepsilon)M_t$, 则由 $\varepsilon$=1/70 可知(1-$\varepsilon$)>1/[2(1-$\varepsilon$)], 进而 $a_{t+1} > \frac{1}{2(1-\varepsilon)} M_t$。由引理

11.10 可知，算法 OnMinSpan 处理完第 $t$ 个任务后，每台机器至少处理一个长度至少为 $\frac{1}{2(1-\varepsilon)}M_t$ 的任务，这意味着最短反例 $I=\langle a_1,a_2,...,a_{t+1}\rangle$ 中至少存在 $m$+1 个长度至少为 $\frac{1}{2(1-\varepsilon)}M_t$ 的任务。由于只有 $m$ 台机器，$I=\langle a_1,a_2,...,a_{t+1}\rangle$ 的最优并行调度中必然存在一台机器至少处理两个长度至少为 $\frac{1}{2(1-\varepsilon)}M_t$ 的任务，即 $OPT_{t+1}\geq\frac{1}{(1-\varepsilon)}M_t$。进而

$$\begin{aligned}On_{t+1}&=M_t+a_{t+1} && \text{（引理 11.8）}\\ &\leq(1-\varepsilon)OPT_{t+1}+a_{t+1}\\ &\leq(1-\varepsilon)OPT_{t+1}+OPT_{t+1} && (OPT_{t+1}\geq M_{t+1})\\ &\leq(2-\varepsilon)OPT_{t+1}\end{aligned}$$

这与$=\langle a_1,a_2,...,a_{t+1}\rangle$ 是算法 OnMinSpan 的(2-$\varepsilon$)-竞争反例矛盾。 □

## 习题

11.1 （滑雪板租买问题）假设您到亚布力滑雪，滑雪板的销售价格是$m$元，租借滑雪板的价格是每天$r$元。由于某些原因，例如是否因为天气恶劣或摔伤而无法滑雪，您无法预知要滑雪的天数。

(1)试设计一个在线算法帮助您每天决定是否在当天买下滑雪板，使得您的实际花销与（滑雪天数事先知道的前提下）的最优花销的比值不超过$m/r$。

(2)设$m$=500,$r$=100,试设计一个竞争度为3的在线算法？

(3)在$m$=500,$r$=100的条件下，在线算法能够获得的最优竞争度是多少？

11.2 现有两个求解计算问题$P$的在线算法$A$和$B$,它们的竞争度分别为2和3,请利用算法$A$和算法$B$设计一个竞争度为9/4的在线随机算法。

11.3 在线最远点对问题要求输出欧几里德平面上在线到来的$n$个点$P_1,...,P_n$中距离最远的两个点。试利用11.3节类似的方法设计一个在线算法求解最远点对问题，并分析算法的竞争度。

11.4 在线1-圆心问题要求输出欧几里德平面上的一个半径最小的圆使得在线到来的$n$个点$P_1,...,P_n$全部位于圆内或边界上。试利用11.3节类似的方法设计一个在线算法求解1-圆心问题，并分析算法的竞争度。

11.5 考虑如下单人记忆游戏。将$n$对扑克牌扣于桌面上，每对扑克牌是相同的。游戏的一个步骤是翻开两张扑克牌，如果它们是相同的牌，则从桌面移除这两张扑克牌；否则将翻开的两张扑克牌重新扣于桌面上。游戏的目的是使用最少的操作步骤移除桌面的2$n$张扑克牌。显然，游戏对于记忆力超强的人而言最少需要$n$个操作步骤。

(1)试设计一个在线算法完成游戏，使得算法的竞争度为2。

(2)试设计一个在线算法完成游戏，使得算法至多需要2$n$-1次操作。

(3)证明：完成记忆游戏的任意在线算法至少需要2$n$-1次操作。

11.6 河边有一堆干草和一头奶牛，奶牛不知道干草堆的方向和距离。下面的在线算法可以帮助奶牛吃到干草。先向左侧移动一个长度单位，如果未发现干草堆则返回出发点；然后向右侧移动两个长度单位，如果未发现干草则返回出发点；再向左移动两个长度单位，...。试构造一个反例，证明上述离线算法的竞争度不小于9。