ТЕОРИЯ РАСПИСАНИЙ

ЛЕКЦИЯ 2

Виктор Васильевич Лепин

Составление временны́х таблиц (Time Tabling)

- Пример — учебное расписание в школе или в ВУЗе.

Составление временны́х таблиц (Time Tabling)

- Пример — учебное расписание в школе или в ВУЗе.
- Чаще всего учебное расписание для группы студентов представляется в виде таблицы (поэтому этот раздел ТР называется “Time Tabling”).

Составление временны́х таблиц (Time Tabling)

- Пример — учебное расписание в школе или в ВУЗе.
- Чаще всего учебное расписание для группы студентов представляется в виде таблицы (поэтому этот раздел ТР называется “Time Tabling”).
- В таблице напересечении строк (дни недели, время лекций) и столбцов (номер группы) указан предмет и номер аудитории, в которой состоится занятие по этому предмету.

Составление временны́х таблиц (Time Tabling)

- Пример — учебное расписание в школе или в ВУЗе.
- Чаще всего учебное расписание для группы студентов представляется в виде таблицы (поэтому этот раздел ТР называется “Time Tabling”).
- В таблице напересечении строк (дни недели, время лекций) и столбцов (номер группы) указан предмет и номер аудитории, в которой состоится занятие по этому предмету.
- Общее расписание ВУЗа — совокупность расписаний для каждой группы.

Составление временны́х таблиц (Time Tabling)

- Пример — учебное расписание в школе или в ВУЗе.
- Чаще всего учебное расписание для группы студентов представляется в виде таблицы (поэтому этот раздел ТР называется “Time Tabling”).
- В таблице напересечении строк (дни недели, время лекций) и столбцов (номер группы) указан предмет и номер аудитории, в которой состоится занятие по этому предмету.
- Общее расписание ВУЗа — совокупность расписаний для каждой группы.
- Фактически в таком расписании согласованы между собой во времени аудитории, группы учащихся и преподаватели.

Составление временны́х таблиц (Time Tabling)

При составлении расписания нужно учесть разнообразные требования, например:

- Условия, связанные с аудиториями: аудитория должна вмещать всех учеников, и в ней должно быть соответствующие оборудование. Единовременно в аудитории может проходить только одно занятие;

Составление временны́х таблиц (Time Tabling)

При составлении расписания нужно учесть разнообразные требования, например:

- Условия, связанные с аудиториями: аудитория должна вмещать всех учеников, и в ней должно быть соответствующие оборудование. Единовременно в аудитории может проходить только одно занятие;
- Условия, связанные со студентами. Желательно, чтобы между лекциями не было больших перерывов. Необходимо чтобы студент успел перейти в другой учебный корпус, если занятия проходят в разных зданиях;

Составление временны́х таблиц (Time Tabling)

При составлении расписания нужно учесть разнообразные требования, например:

- Условия, связанные с аудиториями: аудитория должна вмещать всех учеников, и в ней должно быть соответствующие оборудование. Единовременно в аудитории может проходить только одно занятие;
- Условия, связанные со студентами. Желательно, чтобы между лекциями не было больших перерывов. Необходимо чтобы студент успел перейти в другой учебный корпус, если занятия проходят в разных зданиях;
- Условия, связанные с преподавателями. У преподавателей также есть свои личные предпочтения, например, в какие дни и время проводить занятия;

Составление временны́х таблиц (Time Tabling)

При составлении расписания нужно учесть разнообразные требования, например:

- Условия, связанные с аудиториями: аудитория должна вмещать всех учеников, и в ней должно быть соответствующие оборудование. Единовременно в аудитории может проходить только одно занятие;
- Условия, связанные со студентами. Желательно, чтобы между лекциями не было больших перерывов. Необходимо чтобы студент успел перейти в другой учебный корпус, если занятия проходят в разных зданиях;
- Условия, связанные с преподавателями. У преподавателей также есть свои личные предпочтения, например, в какие дни и время проводить занятия;
- Условия, предъявляемые к учебному процессу. Желательно, чтобы после занятий по физкультуре не было лекционных занятий.

Составление временны́х таблиц (Time Tabling)

- Задачи Time Tabling возникают при планировании занятости персонала, при согласовании времени различных встреч и т.д.

Составление временны́х таблиц (Time Tabling)

- Задачи Time Tabling возникают при планировании занятости персонала, при согласовании времени различных встреч и т.д.
- Зачастую задачи Time Tabling можно свести к задачам Project Scheduling.

Одноприборные задачи TP

Одноприборные задачи TP

- Отметим, что одноприборные задачи являются частными случаями и подзадачами более сложных практических задач.

- Отметим, что одноприборные задачи являются частными случаями и подзадачами более сложных практических задач.
- Для одноприборных задач можно выделить следующее важное свойство.

Теорема 1.

Если моменты поступления требований на обслуживание $r_j = 0$ для всех $j = 1, 2, \ldots, n$, и целевая функция $F(C_1, C_2, \ldots, C_n)$ является монотонно неубывающей функцией, зависящей от моментов окончания обслуживания требований C_j, то в задаче минимизации функции F существует оптимальное расписание без прерываний обслуживания требований и простоев прибора.

Одноприборные задачи TP

- Многие известные нам целевые функции являются монотонно возрастающими (неубывающими) функциями, зависящими от моментов окончания обслуживания требований C_j. Например, $\sum C_j$, $\sum U_j$, $\sum T_j$ и т.д.

Одноприборные задачи TP

- Многие известные нам целевые функции являются монотонно возрастающими (неубывающими) функциями, зависящими от моментов окончания обслуживания требований C_j. Например, $\sum C_j$, $\sum U_j$, $\sum T_j$ и т.д.

Утверждение

Если задача соответствует условиям теоремы 1, тогда оптимальное расписание для этой задачи однозначно задается перестановкой элементов множества N.

Одноприборные задачи TP

- Многие известные нам целевые функции являются монотонно возрастающими (неубывающими) функциями, зависящими от моментов окончания обслуживания требований C_j. Например, $\sum C_j$, $\sum U_j$, $\sum T_j$ и т.д.

Утверждение

Если задача соответствует условиям теоремы 1, тогда оптимальное расписание для этой задачи однозначно задается перестановкой элементов множества N.

Определение

Перестановка из n элементов — это конечная последовательность длины n, все элементы которой различны.

Одноприборные задачи TP

Перестановка вида $\pi = (j_1, j_2, \ldots, j_n)$, задающая расписание π, определяет порядок обслуживания требований.

Одноприборные задачи TP

Перестановка вида $\pi = (j_1, j_2, \ldots, j_n)$, задающая расписание π, определяет порядок обслуживания требований.
Для задач, в которых расписание можно задать перестановкой, важными являются следующие определения:

Перестановка вида $\pi = (j_1, j_2, \ldots, j_n)$, задающая расписание π, определяет порядок обслуживания требований.
Для задач, в которых расписание можно задать перестановкой, важными являются следующие определения:

Определение

EDD (Earliest Due Date) порядок обслуживания требований — очередность обслуживания, при которой требования обслуживаются в порядке неубывания директивных сроков d_j.

Перестановка вида $\pi = (j_1, j_2, \ldots, j_n)$, задающая расписание π, определяет порядок обслуживания требований.
Для задач, в которых расписание можно задать перестановкой, важными являются следующие определения:

Определение

EDD (Earliest Due Date) порядок обслуживания требований — очередность обслуживания, при которой требования обслуживаются в порядке неубывания директивных сроков d_j.

Определение

LDD (Latest Due Date) порядок обслуживания требований — очередность обслуживания, при которой требования обслуживаются в порядке невозрастания директивных сроков d_j.

Определение

SPT (Shortest Processing Time) порядок обслуживания требований — очередность обслуживания, при которой требования обслуживаются в порядке неубывания времен обслуживания p_j.

Определение

SPT (Shortest Processing Time) порядок обслуживания требований — очередность обслуживания, при которой требования обслуживаются в порядке неубывания времен обслуживания p_j.

Определение

LPT (Longest Processing Time) порядок обслуживания требований — очередность обслуживания, при которой требования обслуживаются в порядке невозрастания времен обслуживания p_j.

Определение

SPT (Shortest Processing Time) порядок обслуживания требований — очередность обслуживания, при которой требования обслуживаются в порядке неубывания времен обслуживания p_j.

Определение

LPT (Longest Processing Time) порядок обслуживания требований — очередность обслуживания, при которой требования обслуживаются в порядке невозрастания времен обслуживания p_j.

Определение

Частичное расписание π — фрагмент целого расписания π, описывающее порядок обслуживания подмножества требований $N' \subset N$.

Удобная запись

В данном разделе:

- запись вида $\{\pi\}$ обозначает множество требований, обслуживаемых при расписании π;

Удобная запись

В данном разделе:

- запись вида $\{\pi\}$ обозначает множество требований, обслуживаемых при расписании π;
- запись вида $i \in \pi$ означает $i \in \{\pi\}$;

Удобная запись

В данном разделе:

- запись вида $\{\pi\}$ обозначает множество требований, обслуживаемых при расписании π;
- запись вида $i \in \pi$ означает $i \in \{\pi\}$;
- через $\pi \setminus \{i\}$, где $\pi = (\pi_1, i, \pi_2)$, будем обозначать частичное расписание вида (π_1, π_2);

Удобная запись

В данном разделе:

- запись вида $\{\pi\}$ обозначает множество требований, обслуживаемых при расписании π;
- запись вида $i \in \pi$ означает $i \in \{\pi\}$;
- через $\pi \setminus \{i\}$, где $\pi = (\pi_1, i, \pi_2)$, будем обозначать частичное расписание вида (π_1, π_2);
- запись $j \to i$ означает, что обслуживание требования j предшествует обслуживанию требования i;

Удобная запись

В данном разделе:

- запись вида $\{\pi\}$ обозначает множество требований, обслуживаемых при расписании π;
- запись вида $i \in \pi$ означает $i \in \{\pi\}$;
- через $\pi \setminus \{i\}$, где $\pi = (\pi_1, i, \pi_2)$, будем обозначать частичное расписание вида (π_1, π_2);
- запись $j \to i$ означает, что обслуживание требования j предшествует обслуживанию требования i;
- соответственно, запись $(j \to i)\pi$ означает, что это выполняется при расписании π.

Одноприборные задачи $1|r_j, p_j = 1, pmtn|\sum f_j$

- Рассмотрим алгоритм решения одноприборных задач, для которых продолжительность обслуживания $p_j = 1$, момент поступления $r_j \in \mathbb{Z}^+$, для всех $j = 1, 2, \ldots, n$, и нет отношений предшествования между требованиями.

Одноприборные задачи $1|r_j, p_j = 1, pmtn|\sum f_j$

- Рассмотрим алгоритм решения одноприборных задач, для которых продолжительность обслуживания $p_j = 1$, момент поступления $r_j \in \mathbb{Z}^+$, для всех $j = 1, 2, \ldots, n$, и нет отношений предшествования между требованиями.
- Обозначим эти задачи как

$$1|r_j, p_j = 1, pmtn|\sum f_j$$

При этом функция f_j требования j зависит от времени окончания обслуживания C_j.

Одноприборные задачи $1|r_j, p_j = 1, pmtn|\sum f_j$

- Рассмотрим алгоритм решения одноприборных задач, для которых продолжительность обслуживания $p_j = 1$, момент поступления $r_j \in \mathbb{Z}^+$, для всех $j = 1, 2, \ldots, n$, и нет отношений предшествования между требованиями.
- Обозначим эти задачи как

$$1|r_j, p_j = 1, pmtn|\sum f_j$$

  При этом функция f_j требования j зависит от времени окончания обслуживания C_j.
- Данную задачу можно решить, сведя ее к ЗАДАЧЕ О НАЗНАЧЕНИЯХ.

- Количество заданий в соответствующей ЗАДАЧЕ О НАЗНАЧЕНИЯХ будет равно n, а количество исполнителей равно $r_{\max} + n$, где $r_{\max} \stackrel{def}{=} \max_{j \in \{1,\ldots,n\}} r_j$ – самый поздний момент поступления.

- Количество заданий в соответствующей ЗАДАЧЕ О НАЗНАЧЕНИЯХ будет равно n, а количество исполнителей равно $r_{\max} + n$, где $r_{\max} \stackrel{def}{=} \max_{j \in \{1,\ldots,n\}} r_j$ – самый поздний момент поступления.
- То есть для каждого требования j, $j = 1, 2, \ldots, n$, нужно определить интервал обслуживания $[t, t+1) \in [0, r_{\max} + n)$.

- Количество заданий в соответствующей ЗАДАЧЕ О НАЗНАЧЕНИЯХ будет равно n, а количество исполнителей равно $r_{\max} + n$, где $r_{\max} \stackrel{def}{=} \max_{j \in \{1,\ldots,n\}} r_j$ – самый поздний момент поступления.
- То есть для каждого требования j, $j = 1, 2, \ldots, n$, нужно определить интервал обслуживания $[t, t+1) \in [0, r_{\max} + n)$.
- Матрица стоимостей формируется следующим образом.

- Количество заданий в соответствующей ЗАДАЧЕ О НАЗНАЧЕНИЯХ будет равно n, а количество исполнителей равно $r_{\max} + n$, где $r_{\max} \stackrel{def}{=} \max_{j \in \{1,\ldots,n\}} r_j$ – самый поздний момент поступления.
- То есть для каждого требования j, $j = 1, 2, \ldots, n$, нужно определить интервал обслуживания $[t, t+1) \in [0, r_{\max} + n)$.
- Матрица стоимостей формируется следующим образом.
- Для требования j для каждого $t \in [0, r_{\max} + n)$, $t \geq r_j$, вычислим $a_{tj} = f_j(C_j = t + 1)$.

- Количество заданий в соответствующей ЗАДАЧЕ О НАЗНАЧЕНИЯХ будет равно n, а количество исполнителей равно $r_{\max} + n$, где $r_{\max} \stackrel{def}{=} \max_{j\in\{1,\ldots,n\}} r_j$ – самый поздний момент поступления.
- То есть для каждого требования j, $j = 1, 2, \ldots, n$, нужно определить интервал обслуживания $[t, t+1) \in [0, r_{\max} + n)$.
- Матрица стоимостей формируется следующим образом.
- Для требования j для каждого $t \in [0, r_{\max} + n)$, $t \geq r_j$, вычислим $a_{tj} = f_j(C_j = t + 1)$.
- Для каждой точки $t < r_j$ и для каждой точки $t \geq D_j = r_{\max} + n$ примем $a_{tj} = +\infty$.

- Решив ЗАДАЧУ О НАЗНАЧЕНИЯХ, получим интервалы обслуживания каждого из требований.

- Решив ЗАДАЧУ О НАЗНАЧЕНИЯХ, получим интервалы обслуживания каждого из требований.
- То есть задачу $1|r_j, p_j = 1, pmtn|\sum f_j$ можно решить за время $O(n^3)$.

Минимизация числа запаздывающих требований $1||\sum U_j$

- Имеется множество требований $N = \{1, 2, \ldots, n\}$.

Минимизация числа запаздывающих требований $1||\sum U_j$

- Имеется множество требований $N = \{1, 2, \ldots, n\}$.
- Для каждого требования $j \in N$ заданы продолжительность обслуживания $p_j > 0$ и директивный срок d_j, к которому требование должно быть обслужено.

Минимизация числа запаздывающих требований $1||\sum U_j$

- Имеется множество требований $N = \{1, 2, \ldots, n\}$.
- Для каждого требования $j \in N$ заданы продолжительность обслуживания $p_j > 0$ и директивный срок d_j, к которому требование должно быть обслужено.
- Прерывания при обслуживании и обслуживание более одного требования в любой момент времени запрещены.

Минимизация числа запаздывающих требований $1||\sum U_j$

- Имеется множество требований $N = \{1, 2, \ldots, n\}$.
- Для каждого требования $j \in N$ заданы продолжительность обслуживания $p_j > 0$ и директивный срок d_j, к которому требование должно быть обслужено.
- Прерывания при обслуживании и обслуживание более одного требования в любой момент времени запрещены.
- Расписание задается перестановкой $\pi = (j_1, j_2, \ldots, j_n)$.

Минимизация числа запаздывающих требований $1||\sum U_j$

- Имеется множество требований $N = \{1, 2, \ldots, n\}$.
- Для каждого требования $j \in N$ заданы продолжительность обслуживания $p_j > 0$ и директивный срок d_j, к которому требование должно быть обслужено.
- Прерывания при обслуживании и обслуживание более одного требования в любой момент времени запрещены.
- Расписание задается перестановкой $\pi = (j_1, j_2, \ldots, j_n)$.
- То есть время завершения обслуживания требования j_k при расписании π определяется следующим образом: $C_{jk}(\pi) = \sum_{l=1}^{k} p_{jl}$.

Минимизация числа запаздывающих требований $1||\sum U_j$

- Имеется множество требований $N = \{1, 2, \ldots, n\}$.
- Для каждого требования $j \in N$ заданы продолжительность обслуживания $p_j > 0$ и директивный срок d_j, к которому требование должно быть обслужено.
- Прерывания при обслуживании и обслуживание более одного требования в любой момент времени запрещены.
- Расписание задается перестановкой $\pi = (j_1, j_2, \ldots, j_n)$.
- То есть время завершения обслуживания требования j_k при расписании π определяется следующим образом: $C_{jk}(\pi) = \sum_{l=1}^{k} p_{jl}$.
- Если $C_j(\pi) > d_j$, тогда требование j запаздывает, и в этом случае полагают $U_j = 1$.

Минимизация числа запаздывающих требований $1||\sum U_j$

- Имеется множество требований $N = \{1, 2, \ldots, n\}$.
- Для каждого требования $j \in N$ заданы продолжительность обслуживания $p_j > 0$ и директивный срок d_j, к которому требование должно быть обслужено.
- Прерывания при обслуживании и обслуживание более одного требования в любой момент времени запрещены.
- Расписание задается перестановкой $\pi = (j_1, j_2, \ldots, j_n)$.
- То есть время завершения обслуживания требования j_k при расписании π определяется следующим образом: $C_{jk}(\pi) = \sum_{l=1}^{k} p_{jl}$.
- Если $C_j(\pi) > d_j$, тогда требование j запаздывает, и в этом случае полагают $U_j = 1$.
- Если $C_j(\pi) \leq d_j$, тогда требование j не запаздывает, и $U_j = 0$.

Минимизация числа запаздывающих требований $1||\sum U_j$

- Имеется множество требований $N = \{1, 2, \ldots, n\}$.
- Для каждого требования $j \in N$ заданы продолжительность обслуживания $p_j > 0$ и директивный срок d_j, к которому требование должно быть обслужено.
- Прерывания при обслуживании и обслуживание более одного требования в любой момент времени запрещены.
- Расписание задается перестановкой $\pi = (j_1, j_2, \ldots, j_n)$.
- То есть время завершения обслуживания требования j_k при расписании π определяется следующим образом: $C_{jk}(\pi) = \sum_{l=1}^{k} p_{jl}$.
- Если $C_j(\pi) > d_j$, тогда требование j запаздывает, и в этом случае полагают $U_j = 1$.
- Если $C_j(\pi) \leq d_j$, тогда требование j не запаздывает, и $U_j = 0$.
- Необходимо построить расписание π, при котором значение целевой функции $F(\pi) = \sum_{j=1}^{n} U_j(\pi)$ минимально.

Минимизация числа запаздывающих требований $1||\sum U_j$

Далее представлен полиномиальный алгоритм Мура решения данной задачи, который основан на следующей лемме.

Минимизация числа запаздывающих требований $1||\sum U_j$

Далее представлен полиномиальный алгоритм Мура решения данной задачи, который основан на следующей лемме.

Лемма 1

Для каждого примера задачи $1||\sum U_j$ существует оптимальное расписание вида $\pi = (G, H)$, при котором все требования $i \in G$ не запаздывают, а все требования $j \in H$ запаздывают.

Минимизация числа запаздывающих требований $1||\sum U_j$

Далее представлен полиномиальный алгоритм Мура решения данной задачи, который основан на следующей лемме.

Лемма 1

Для каждого примера задачи $1||\sum U_j$ существует оптимальное расписание вида $\pi = (G, H)$, при котором все требования $i \in G$ не запаздывают, а все требования $j \in H$ запаздывают.

Лемма 2

Пусть для примера задачи $1||\sum U_j$ существует оптимальное расписание, при котором все требования не запаздывают. Тогда расписание π_{EDD} (требования обслуживаются в порядке EDD) также является оптимальным для данного примера.

Algorithm 1

Перенумеруем требования согласно правилу $d_1 \le d_2 \le \cdots \le d_n$;
$G := \emptyset$; $H := \emptyset$; $t := 0$;
for $j = 1$ to n **do**
  $G \leftarrow G \cup \{j\}$;
  $t \leftarrow t + p_j$;
  if $t \ge d_j$ **then**
    Найдем требование $i \in G$ с максимальной продолжительность обслуживания, т.е.
    $i := \arg\max_{k \in G} p_k$;
    $G \leftarrow G \setminus \{i\}$;
    $H \leftarrow H \cup \{i\}$;
    $t \leftarrow t - p_i$;
  end if
end for
return $\pi = (G, H)$.

Лемма 3

Пусть при расписании $\pi = (j_1, j_2, \ldots, j_l, j_{l+1}, \ldots, j_n)$ *выполняется* $d_{j_1} \leq d_{j_2} \leq \cdots d_{j_n}$. *Требование* j_l — *последнее незапаздывающее требование. Тогда существует оптимальное расписание, при котором требование* $j^* \in \{j_1, j_2, \ldots, j_{l+1}\}$, $p_{j^*} \geq p_i$, *для всех* $i \in \{j_1, j_2, \ldots, j_{l+1}\}$, *запаздывает.*

Алгоритм решения задачи $1||\sum U_j$

Лемма 3

Пусть при расписании $\pi = (j_1, j_2, \ldots, j_l, j_{l+1}, \ldots, j_n)$ *выполняется* $d_{j_1} \leq d_{j_2} \leq \cdots d_{j_n}$. *Требование* j_l — *последнее незапаздывающее требование. Тогда существует оптимальное расписание, при котором требование* $j^* \in \{j_1, j_2, \ldots, j_{l+1}\}$, $p_{j^*} \geq p_i$, *для всех* $i \in \{j_1, j_2, \ldots, j_{l+1}\}$, *запаздывает.*

Теорема 2

Алгоритм 1 для задачи $1||\sum U_j$ *за* $O(n \log n)$ *операций строит оптимальное расписание.*

Минимизация взвешенного числа запаздывающих требований $1||\sum w_j U_j$

Очевидным является тот факт, что минимизация взвешенного числа запаздывающих требований эквивалентно максимизации взвешенного числа незапаздывающих требований. То есть критерий оптимизации $F(\pi) = \sum w_j U_j(\pi) \to \min$ может быть заменен на критерий $F(\pi) = \sum w_j [1 - U_j(\pi)] \to \max$.

Минимизация взвешенного числа запаздывающих требований $1||\sum w_jU_j$

Очевидным является тот факт, что минимизация взвешенного числа запаздывающих требований эквивалентно максимизации взвешенного числа незапаздывающих требований. То есть критерий оптимизации $F(\pi) = \sum w_jU_j(\pi) \to \min$ может быть заменен на критерий $F(\pi) = \sum w_j[1 - U_j(\pi)] \to \max$.

Теорема 3

Задача $1||\sum w_jU_j$ является NP-трудной.

Минимизация взвешенного числа запаздывающих требований $1||\sum w_jU_j$

Очевидным является тот факт, что минимизация взвешенного числа запаздывающих требований эквивалентно максимизации взвешенного числа незапаздывающих требований. То есть критерий оптимизации $F(\pi) = \sum w_jU_j(\pi) \to \min$ может быть заменен на критерий $F(\pi) = \sum w_j[1 - U_j(\pi)] \to \max$.

Теорема 3

Задача $1||\sum w_jU_j$ является NP-трудной.

Теорема 4

Для задачи $1||\sum w_jU_j$ существует оптимальное расписание вида $\pi = (G, H)$, где все требования $j \in H$ запаздывают, а все требования $i \in G$ не запаздывают. Требования из множества G обслуживаются в порядке EDD, а требования из множества H — в порядке LDD.

Алгоритм динамического программирования для решения задачи $1||\sum w_j U_j$

Algorithm 2

Перенумеруем требования согласно правилу $d_1 \leq d_2 \leq \cdots \leq d_n$;
for $t = 0$ to $\sum_{i=2}^{n} p_i$ **do**
  $\pi_1(t) := (1)$;
  if $t + p_1 - d_1 \leq 0$ **then**
    $f_1(t) := w_1$;
  else
    $f_1(t) := 0$;
  end if
end for

for $j = 2$ to n **do**
  for $t = 0$ to $\sum_{i=j+1}^{n} p_i$ **do**
    $\pi^1 := (j, \pi_{j-1}(t + p_j))$, $\pi^2 := (\pi_{j-1}(t), j)$;
    if $t + p_j - d_j \leq 0$ **then**
      $\Phi^1(t) := w_j + f_{j-1}(t + p_j)$;
    else
      $\Phi^1(t) := f_{j-1}(t + p_j)$;
    end if
    if $t + \sum_{i=1}^{j} p_i - d_j \leq 0$ **then**
      $\Phi^2(t) := f_{j-1}(t) + w_j$;
    else
      $\Phi^2(t) := f_{j-1}(t)$;
    end if
    if $\Phi^1(t) < \Phi^2(t)$ **then**
      $f_j(t) := \Phi^2(t)$;
      $\pi_j(t) := \pi^2$;
    else
      $f_j(t) := \Phi^1(t)$;
      $\pi_j(t) := \pi^1$;
    end if
  end for
end for

Алгоритм 2 основан на теореме 4.
В данном алгоритме $\pi_j(t)$ означает оптимальное частичное расписание для требований $1, 2, \ldots, j$, при котором обслуживание требований начинается в момент времени t, а $f_j(t) = \sum_{i=1}^{j} w_i[1 - U_i(\pi_j(t))]$ соответствует максимальному взвешенному числу незапаздывающих требований при этом частичном расписании.

Алгоритм 2 основан на теореме 4.
В данном алгоритме $\pi_j(t)$ означает оптимальное частичное расписание для требований $1, 2, \ldots, j$, при котором обслуживание требований начинается в момент времени t, а $f_j(t) = \sum_{i=1}^{j} w_i[1 - U_i(\pi_j(t))]$ соответствует максимальному взвешенному числу незапаздывающих требований при этом частичном расписании.
В Алгоритме 2 происходит последовательная "окантовка" ранее построенного расписания. В расписании $\pi_l(t)$ для любого l, $l = 1, 2, \ldots, n$, требования множества $\{1, 2, \ldots, l\}$ расположены "компактно", т.е. между ними нет требований с большими номерами.

Алгоритм 2 основан на теореме 4.
В данном алгоритме $\pi_j(t)$ означает оптимальное частичное расписание для требований $1, 2, \ldots, j$, при котором обслуживание требований начинается в момент времени t, а $f_j(t) = \sum_{i=1}^{j} w_i[1 - U_i(\pi_j(t))]$ соответствует максимальному взвешенному числу незапаздывающих требований при этом частичном расписании.
В Алгоритме 2 происходит последовательная "окантовка" ранее построенного расписания. В расписании $\pi_l(t)$ для любого l, $l = 1, 2, \ldots, n$, требования множества $\{1, 2, \ldots, l\}$ расположены "компактно", т.е. между ними нет требований с большими номерами.

Теорема 5

Алгоритм 2 строит оптимальное расписание задачи максимизации взвешенного числа незапаздывающих требований за $O(n \sum p_j)$ операций.

Минимизация суммарного запаздывания $1||\sum T_j$

- Рассматривается задача минимизации суммарного запаздывания $\sum T_j$, где $T_j = \max\{0, C_j - d_j\}$.

Минимизация суммарного запаздывания $1||\sum T_j$

- Рассматривается задача минимизации суммарного запаздывания $\sum T_j$, где $T_j = \max\{0, C_j - d_j\}$.
- То есть в данной задаче необходимо найти расписание π^*, при котором функция $F(\pi) = \sum_{j=1}^{n} T_j(\pi)$ достигает своего минимума.

Минимизация суммарного запаздывания $1||\sum T_j$

- Рассматривается задача минимизации суммарного запаздывания $\sum T_j$, где $T_j = \max\{0, C_j - d_j\}$.
- То есть в данной задаче необходимо найти расписание π^*, при котором функция $F(\pi) = \sum_{j=1}^{n} T_j(\pi)$ достигает своего минимума.
- Несмотря на то, что это классическая задача ТР, доказательство ее NP-трудности было получено сравнительно недавно — в 1990-м году, т.е. выяснение ее трудоемкости было нетривиальной проблемой.

Точный алгоритм решения задачи $1||\sum T_j$

- Алгоритм основан на результатах, опубликованных Лаулером в 1977-м году.
- Необходимо отметить, что данный алгоритм строит точное решение и для частного случая задачи $1||\sum w_j T_j$, при котором выполняется правило “если $p_j > p_i$, то $w_j \leq w_i$, $i, j \in N$”.

Теорема 6

Пусть π *— оптимальное расписание для примера задачи с директивными сроками* $d_1, d_2, \ldots, d_n$, *и пусть* C_j *— моменты завершения обслуживания требований* $j = 1, 2, \ldots, n$ *при этом расписании. Выберем новые директивные сроки* d'_j *так, что:*

$$\min\{d_j, C_j\} \leq d'_j \leq \max\{d_j, C_j\}.$$

Тогда любое оптимальное расписание π', *соответствующее примеру с новыми директивными сроками* $d'_1, d'_2, \ldots, d'_n$, *является оптимальным и для примера с исходными директивными сроками* $d_1, d_2, \ldots, d_n$.

Алгоритм решения задачи $1||\sum T_j$

- Можно сделать следующий вывод из данной теоремы.

Алгоритм решения задачи $1||\sum T_j$

- Можно сделать следующий вывод из данной теоремы.
- Если выбрать такое требование j^* и для него определить требование (позицию) $k \geq j^*$, то исходную задачу можно разбить на две подзадачи решаемые аналогично.

- Можно сделать следующий вывод из данной теоремы.
- Если выбрать такое требование j^* и для него определить требование (позицию) $k \geq j^*$, то исходную задачу можно разбить на две подзадачи решаемые аналогично.
- Первая подзадача содержит множество требований i, $i = 1, 2, \ldots, k$, $i \neq j^*$, а другая — множество требований $\{k+1, k+2, \ldots, n\}$.

Алгоритм решения задачи $1||\sum T_j$

- Можно сделать следующий вывод из данной теоремы.
- Если выбрать такое требование j^* и для него определить требование (позицию) $k \geq j^*$, то исходную задачу можно разбить на две подзадачи решаемые аналогично.
- Первая подзадача содержит множество требований i, $i = 1, 2, \ldots, k$, $i \neq j^*$, а другая — множество требований $\{k+1, k+2, \ldots, n\}$.
- Сложность заключается в выборе требования (позиции) k.

Алгоритм решения задачи $1||\sum T_j$

- Можно сделать следующий вывод из данной теоремы.
- Если выбрать такое требование j^* и для него определить требование (позицию) $k \geq j^*$, то исходную задачу можно разбить на две подзадачи решаемые аналогично.
- Первая подзадача содержит множество требований i, $i = 1, 2, \ldots, k$, $i \neq j^*$, а другая — множество требований $\{k+1, k+2, \ldots, n\}$.
- Сложность заключается в выборе требования (позиции) k.
- На этом факте основан следующий точный алгоритм решения задачи.

Algorithm 3

Procedure ProcL(N, t)

Дан пример $\{N, t\}$ с множеством требований $N = \{j_1, j_2, \ldots, j_n\}$ и моментом начала обслуживания t, где $d_{j_1} \le d_{j_2} \le \cdots \le d_{j_n}$.

if $N = \emptyset$ **then**

π^* :=пустое расписание;

else

Найдем требование j^* из множества N;

Найдем множество $L(N, t)$ для требования j^*;

for ALL $k \in L(N, t)$ **do**

$\pi_k := (\mathbf{ProcL}(N', t'), j^*(N), \mathbf{ProcL}(N'', t''))$, где

$N' := \{j_1, \ldots, j_k\} \setminus \{j^*\}$, $t' := t$, $N'' := \{j_{k+1}, \ldots, j_n\}$,

$t'' := t + \sum_{i=1}^{k} p_{j_i}$;

end for

$\pi^* := \arg\min_{k \in L(N,t)} \{F(\pi_k, t)\}$;

end if

return π^*;

Алгоритм решения.

$\pi^* :=$ **ProcL**$(N, 0)$;

Алгоритм решения задачи $1||\sum T_j$

- В алгоритме $j^*(N')$ обозначает требование с наибольшей продолжительностью обслуживания среди требований множества $N' \subseteq N$.

Алгоритм решения задачи $1||\sum T_j$

- В алгоритме $j^*(N')$ обозначает требование с наибольшей продолжительностью обслуживания среди требований множества $N' \subseteq N$.
- Если таких требований несколько, то выбирается требование с наибольшим директивным сроком, т.е. $j^*(N') = \arg\max_{j \in N'}\{d_j \ : \ p_j = \max_{i \in N'} p_i\}$.

Алгоритм решения задачи $1||\sum T_j$

- В алгоритме $j^*(N')$ обозначает требование с наибольшей продолжительностью обслуживания среди требований множества $N' \subseteq N$.
- Если таких требований несколько, то выбирается требование с наибольшим директивным сроком, т.е. $j^*(N') = \arg\max_{j \in N'}\{d_j \ : \ p_j = \max_{i \in N'} p_i\}$.
- Для сокращения записи вместо $j^*(N')$ будем записывать j^*, если очевидно о каком множестве идет речь.

Алгоритм решения задачи $1||\sum T_j$

- В алгоритме $j^*(N')$ обозначает требование с наибольшей продолжительностью обслуживания среди требований множества $N' \subseteq N$.
- Если таких требований несколько, то выбирается требование с наибольшим директивным сроком, т.е. $j^*(N') = \arg\max_{j \in N'}\{d_j \ : \ p_j = \max_{i \in N'} p_i\}$.
- Для сокращения записи вместо $j^*(N')$ будем записывать j^*, если очевидно о каком множестве идет речь.
- Рассмотрим пример (подпример) обслуживания требований множества $N' \subseteq N$, $N' = \{1, 2, \ldots, n'\}$, с момента времени $t' \geq 0$. Множество $L(N', t')$ есть множество всех индексов $i \in \{j^*, j^* + 1, \ldots, n'\}$.

Алгоритм решения задачи $1||\sum T_j$

- В алгоритме $j^*(N')$ обозначает требование с наибольшей продолжительностью обслуживания среди требований множества $N' \subseteq N$.
- Если таких требований несколько, то выбирается требование с наибольшим директивным сроком, т.е. $j^*(N') = \arg\max_{j \in N'}\{d_j \ : \ p_j = \max_{i \in N'} p_i\}$.
- Для сокращения записи вместо $j^*(N')$ будем записывать j^*, если очевидно о каком множестве идет речь.
- Рассмотрим пример (подпример) обслуживания требований множества $N' \subseteq N$, $N' = \{1, 2, \ldots, n'\}$, с момента времени $t' \geq 0$. Множество $L(N', t')$ есть множество всех индексов $i \in \{j^*, j^* + 1, \ldots, n'\}$.
- Запись $F(\pi_k, t)$ в алгоритме означает суммарное запаздывание при расписании π_k выполнение которого начинается с момента времени t.

Задачи цеха (Shop problems)

Задачи $F||C_{\max}$

- В этом разделе рассматривается задача Flow-Shop, в которой необходимо минимизировать значение $C_{\max} = \max_{j \in N} C_j$.

Задачи $F||C_{\max}$

- В этом разделе рассматривается задача Flow-Shop, в которой необходимо минимизировать значение $C_{\max} = \max_{j \in N} C_j$.
- Напомним, что для данной задачи каждое требование j, $j = 1, 2, \ldots, n$, состоит из одних и тех же операций, т.е. $O_{j1} \to \cdots \to O_{jm}$, $\forall j \in N$, причем для каждой операции с номером i, $i = 1, 2, \ldots, m$, задан прибор, выполняющий эту операцию.

Задачи $F||C_{\max}$

- В этом разделе рассматривается задача Flow-Shop, в которой необходимо минимизировать значение $C_{\max} = \max_{j \in N} C_j$.
- Напомним, что для данной задачи каждое требование j, $j = 1, 2, \ldots, n$, состоит из одних и тех же операций, т.е. $O_{j1} \to \cdots \to O_{jm}$, $\forall j \in N$, причем для каждой операции с номером i, $i = 1, 2, \ldots, m$, задан прибор, выполняющий эту операцию.
- Операции каждого требования j выполняются в заданной последовательности $O_{j1} \to \cdots \to O_{jm}$, причем выполнение операции k, $k = 2, 3, \ldots, m$, может начаться не раньше окончания выполнения операции $k - 1$ для этого же требования.

Задачи $F||C_{\max}$

- В этом разделе рассматривается задача Flow-Shop, в которой необходимо минимизировать значение $C_{\max} = \max_{j \in N} C_j$.
- Напомним, что для данной задачи каждое требование j, $j = 1, 2, \ldots, n$, состоит из одних и тех же операций, т.е. $O_{j1} \to \cdots \to O_{jm}$, $\forall j \in N$, причем для каждой операции с номером i, $i = 1, 2, \ldots, m$, задан прибор, выполняющий эту операцию.
- Операции каждого требования j выполняются в заданной последовательности $O_{j1} \to \cdots \to O_{jm}$, причем выполнение операции k, $k = 2, 3, \ldots, m$, может начаться не раньше окончания выполнения операции $k - 1$ для этого же требования.
- Расписание для каждого прибора задается вектором — порядком обслуживания на данном приборе операций, относящихся к разным требованиям.

На рис.3 представлено два допустимых расписания для одного и того же примера, где $n = 2$ и $m = 4$. При этих расписаниях $C_{\max} = 11$.

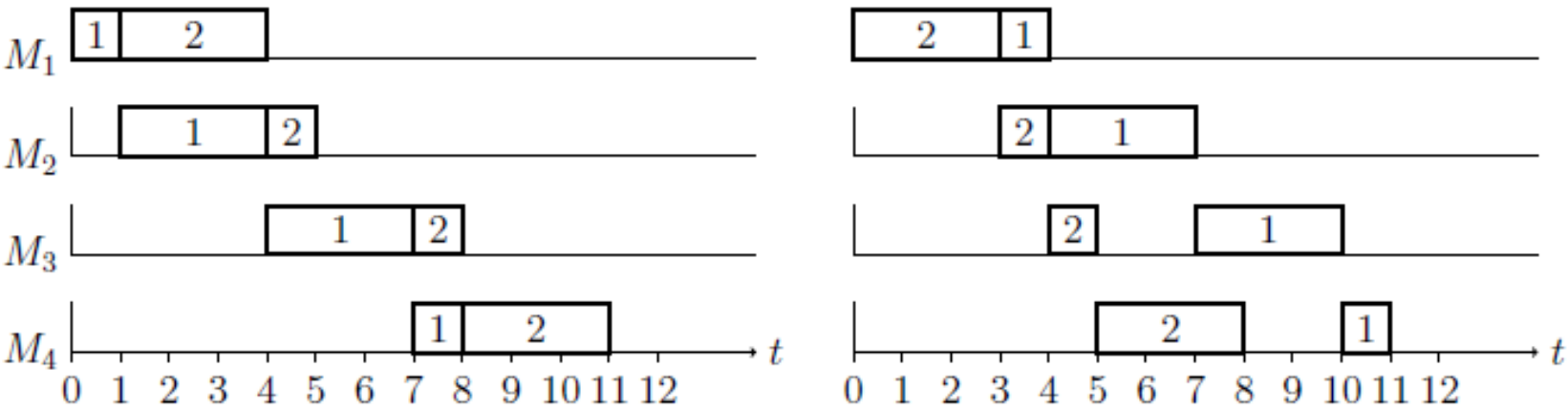

Рис. 3 Расписания для двух требований

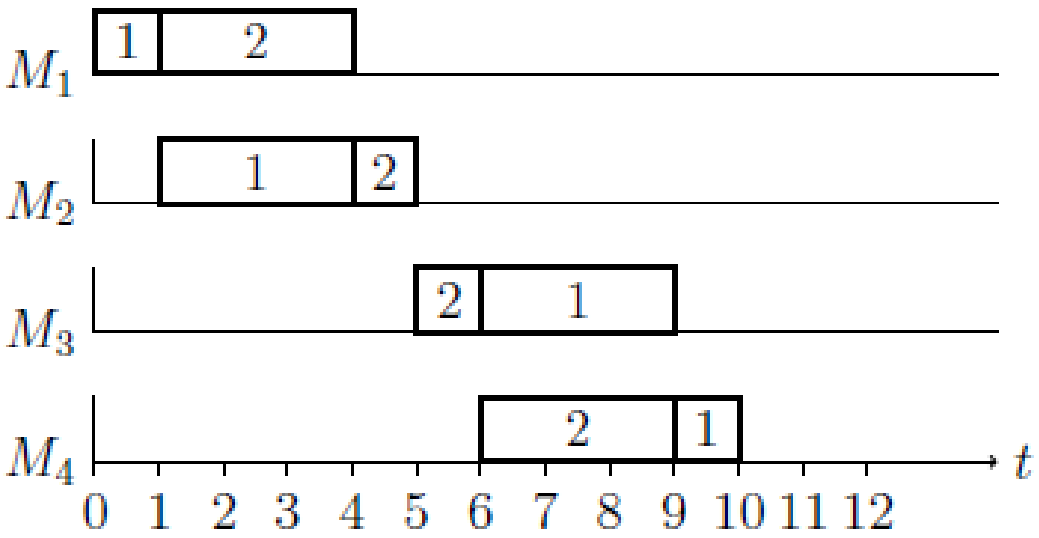

Рис. 4. Оптимальное расписание для двух требований $C_{\max} = 10$.

Теорема 7

Для задачи $F||C_{\max}$ существует оптимальное расписание, обладающее свойствами:

- *порядок ослуживания требований (вектор) для первых двух приборов совпадает;*
- *порядок ослуживания требований (вектор) для последних двух приборов совпадает;*

Теорема 7

Для задачи $F||C_{\max}$ существует оптимальное расписание, обладающее свойствами:

- *порядок ослуживания требований (вектор) для первых двух приборов совпадает;*
- *порядок ослуживания требований (вектор) для последних двух приборов совпадает;*

На основании этой теоремы можно сделать вывод, что если $m \leq 3$, то существует порядок обслуживания, оптимальный (и одинаковый) для всех трех приборов.

Далее представлен алгоритм Джонсона решения задачи $F2||C_{\max}$, полученный в 1954-м году.
Идея алгоритма заключается в следующем.

Далее представлен алгоритм Джонсона решения задачи $F2||C_{\max}$, полученный в 1954-м году.
Идея алгоритма заключается в следующем.

- Последовательно мы конструируем начало π_1 и конец π_2 вектора $\pi = (\pi_1, \pi_2)$, задающего порядок обслуживания.

Далее представлен алгоритм Джонсона решения задачи $F2||C_{\max}$, полученный в 1954-м году.
Идея алгоритма заключается в следующем.

- Последовательно мы конструируем начало π_1 и конец π_2 вектора $\pi = (\pi_1, \pi_2)$, задающего порядок обслуживания.
- На каждом шаге мы рассматриваем операцию O_{ji} с наменьшим значением p_{ji} среди рассматриваемых операций.

Задачи $F||C_{\max}$

Далее представлен алгоритм Джонсона решения задачи $F2||C_{\max}$, полученный в 1954-м году.
Идея алгоритма заключается в следующем.

- Последовательно мы конструируем начало π_1 и конец π_2 вектора $\pi = (\pi_1, \pi_2)$, задающего порядок обслуживания.
- На каждом шаге мы рассматриваем операцию O_{ji} с наменьшим значением p_{ji} среди рассматриваемых операций.
- Если $i = 1$, то в конец частичного расписания π_1 мы добавляем требование j, т.е. $\pi_1 = (\pi_1, j)$, иначе требование добавляется в начало другого частичного расписания $\pi_2 = (j, \pi_2)$.

Далее представлен алгоритм Джонсона решения задачи $F2||C_{\max}$, полученный в 1954-м году.
Идея алгоритма заключается в следующем.

- Последовательно мы конструируем начало π_1 и конец π_2 вектора $\pi = (\pi_1, \pi_2)$, задающего порядок обслуживания.
- На каждом шаге мы рассматриваем операцию O_{ji} с наменьшим значением p_{ji} среди рассматриваемых операций.
- Если $i = 1$, то в конец частичного расписания π_1 мы добавляем требование j, т.е. $\pi_1 = (\pi_1, j)$, иначе требование добавляется в начало другого частичного расписания $\pi_2 = (j, \pi_2)$.
- После этого требование j исключается из рассмотрения и мы повторяем итерацию до тех пор, пока не будут расставлены все требования.

Algorithm 4

$N' := N$; $\pi_1 := ()$, $\pi_2 := ()$;
while $N' \neq \emptyset$ **do**
Найдем $O_{j^*i^*}$ с минимальной продолжительностью
$p_{j^*i^*} = \min\{p_{ji} | j \in N', i = 1, 2\}$;
if $i = 1$ **then**
$\pi_1 := (\pi_1, j)$;
else
$\pi_2 := (j, \pi_2)$;
end if
$N' := N' \setminus j$;
end while
$\pi := (\pi_1, \pi_2)$;
return π;
END.

Необходимо отметить, что алгоритм 4 имеет трудоемкость $O(n \log n)$ операций, если пункт 3 алгоритма — сортировку продолжительности обслуживания требований — выполнить до основного цикла WHILE.

Необходимо отметить, что алгоритм 4 имеет трудоемкость $O(n \log n)$ операций, если пункт 3 алгоритма — сортировку продолжительности обслуживания требований — выполнить до основного цикла WHILE.

Теорема 8

Задача $F3||C_{\max}$ *NP-сложна в сильном смысле.*

Необходимо отметить, что алгоритм 4 имеет трудоемкость $O(n \log n)$ операций, если пункт 3 алгоритма — сортировку продолжительности обслуживания требований — выполнить до основного цикла WHILE.

Теорема 8

Задача $F3||C_{\max}$ *NP-сложна в сильном смысле.*

Задача с прерываниями также является сложнорешаемой.

Теорема 9

Задача $F3|prmp|C_{\max}$ *NP-сложна в сильном смысле.*