

# Задание № 1 **Задание 1. Задача упорядочивания nx2**

1. Разобраться с построением модели задачи Джонсона‐Белмана nx2 из теории расписаний.
2. Ознакомиться с формулировкой задачи Джонсона‐Белмана nx2.
3. Научиться строить график Ганта для задачи Джонсона‐Белмана nx2 (с n деталями и двумя станками).
4. Изучить процесс построения модели задачи Джонсона (nx2) на основе графика Ганта.
5. Изучить возможности поиска оптимального решения задачи Джонсона для двух станков. Изучить подход вывода алгоритма Джонсона и сам алгоритм. Решить задачу упорядочения согласно варианту с использованием алгоритма Джонсона, найдя оптимальный порядок запуска последовательности *n* деталей в обработку на двух станках.
6. Построить график Ганта для оптимальной последовательности запуска деталей своей задачи согласно варианту.
7. Рассчитать оптимальное время окончания обработки всех деталей на двух станках. Это удобно сделать, перенумеровывая оптимальный порядок в порядке возрастания индексов и затем использовать формулы из пункта 4. График Ганта тоже можно приводить с новой нумерацией и новым определением цвета.

Вариант №23

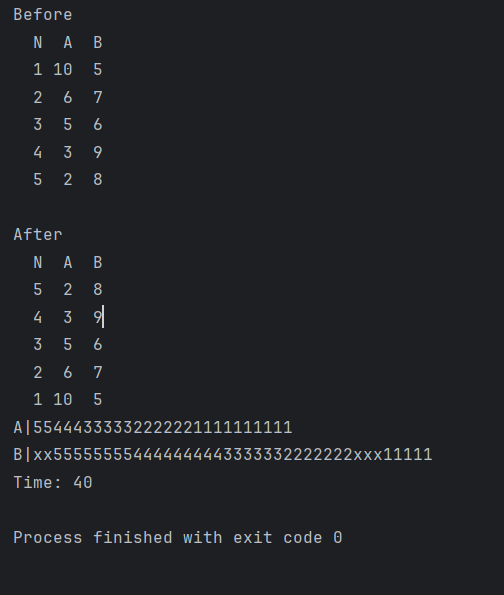
|  |  |  |
| --- | --- | --- |
| i | Ai | bi |
| 1 | 10 | 5 |
| 2 | 6 | 7 |
| 3 | 5 | 6 |
| 4 | 3 | 9 |
| 5 | 2 | 8 |

# **Решение задачи задания 1**

## Алгоритм решения:

1. Находим оптимальную последовательность в задаче упорядочивания nx2, применяя алгоритм Джонсона.
   1. Найти наименьший элемент в таблице значений {ai,bi}.
   2. Запомнить индекс строки таблицы с наименьшим элементом, и добавить его в массив, который хранит оптимальную последовательность
   3. Если этот наименьший элемент принадлежит столбцу значений ai, то деталь с соответствующим номером i поставить на первое свободное место в массив, который соответствует определенному столбцу таблицы,(в первый раз это будет первое место, во второй – второе и т.д.); если же наименьший элемент принадлежит столбцу значений bi, то деталь с соответствующим номером i поставить на последнее свободное место в массив, который соответствует определенному столбцу таблицы (в первый раз это будет n-е место, во второй – (n - 1) и т.д.); если одновременно найдется несколько одинаковых наименьших элементов, то среди них можно выбирать любой, принять за наименьший и поступать так, как описано в начале пункта b алгоритма.
   4. Из таблицы значений {ai,bi} вычеркнуть строку, соответствующую выбранному наименьшему элементу, и проверить, остались ли еще не вычеркнутые строки.
   5. Если не вычеркнутые строки еще остались, то рассматривать их как новую таблицу значений {ai,bi} и перейти к пункту a алгоритма; если вычеркнуты все строки таблицы, то это означает, что алгоритм свою работу закончил.
2. Рассчитываем время обработки
   1. Находим сумму простоев по формуле:
   2. К сумме простоев прибавляем
3. Рисуем график Ганта, используя формулу нахождения простоев на втором станке:

public class Main {  
 public Main() {  
 }  
  
 static void drawDiagram(int[] s, int[] a, int[] b) {  
 System.*out*.print("A|");  
  
 int x;  
 int time;  
 for(x = 0; x < a.length; ++x) {  
 for(time = 0; time < a[x]; ++time) {  
 System.*out*.print(s[x]);  
 }  
 }  
  
 System.*out*.println();  
 System.*out*.print("B|");  
 time = 0;  
  
 for(int i = 0; i < b.length; ++i) {  
 if (i == 0) {  
 x = a[i];  
 } else {  
 x = Math.*max*(a[i] - b[i - 1], 0);  
 }  
  
 time += x;  
  
 int j;  
 for(j = 0; j < x; ++j) {  
 System.*out*.print("x");  
 }  
  
 time += b[i];  
  
 for(j = 0; j < b[i]; ++j) {  
 System.*out*.print(s[i]);  
 }  
 }  
  
 System.*out*.println();  
 System.*out*.println("Time: " + time);  
 }  
  
 static void print(int[] s, int[] a, int[] b) {  
 System.*out*.printf("%3c%3c%3c\n", 'N', 'A', 'B');  
  
 for(int i = 0; i < a.length; ++i) {  
 System.*out*.printf("%3d%3d%3d\n", s[i], a[i], b[i]);  
 }  
  
 }  
  
 static void jhonson(int[] s, int[] a, int[] b) {  
 int aIndex = 0;  
 int bIndex = a.length;  
  
 while(bIndex - aIndex > 1) {  
 int minA = a[aIndex];  
 int minB = b[aIndex];  
 int rowA = aIndex;  
 int rowB = aIndex;  
  
  
 for(int j = aIndex; j < bIndex; ++j) {  
 if (minA > a[j]) {  
 minA = a[j];  
 rowA = j;  
 }  
  
 if (minB > b[j]) {  
 minB = b[j];  
 rowB = j;  
 }  
 }  
  
 if (minA < minB) {  
 int temp = s[aIndex];  
 s[aIndex] = s[rowA];  
 s[rowA] = temp;  
  
 temp = a[aIndex];  
 a[aIndex] = a[rowA];  
 a[rowA] = temp;  
 temp = b[aIndex];  
 b[aIndex] = b[rowA];  
 b[rowA] = temp;  
 ++aIndex;  
  
 } else {  
 int temp = s[bIndex - 1];  
 s[bIndex - 1] = s[rowB];  
 s[rowB] = temp;  
 temp = a[bIndex - 1];  
 a[bIndex - 1] = a[rowB];  
 a[rowB] = temp;  
 temp = b[bIndex - 1];  
 b[bIndex - 1] = b[rowB];  
 b[rowB] = temp;  
 --bIndex;  
 }  
 }  
  
 }  
  
 public static void main(String[] args) {  
 int[] s = new int[]{1, 2, 3, 4, 5};  
 int[] a = new int[]{10, 6, 5, 3, 2};  
 int[] b = new int[]{5, 7, 6, 9, 8};  
 System.*out*.println("Before");  
 *print*(s, a, b);  
 *jhonson*(s, a, b);  
 System.*out*.println("\nAfter");  
 *print*(s, a, b);  
 *drawDiagram*(s, a, b);  
 }  
}



# **Задание 2. Задача упорядочивания nx3**

1. Разобраться с построением модели задачи Джонсона‐ Белмана nx3 из теории расписаний.
2. Ознакомиться с формулировкой задачи Джонсона‐ Белмана nx3.
3. Научиться строить график Ганта для задачи Джонсона‐Белмана с n деталями и тремя станками.
4. Изучить процесс построения модели задачи Джонсона с (nx3) на основе графика Ганта.
5. Изучить возможности поиска оптимального решения задачи Джонсона для трех станков.

# **Решение задачи задания 2**

## Алгоритм решения:

1. Проверяем таблицу nx3 на выполнение условия сводимости к nx2
2. Если условие выполняется, то сводим таблицу nx3 к nx2
   1. Производим построчное сложение элементов первого и второго столбцов таблицы: di=ai+bi .
   2. Производим сложение второго и третьего столбцов таблицы ei=bi+ci.
3. У полученной таблицы находим оптимальную последовательность в задаче упорядочивания nx2, применяя алгоритм Джонсона.
   1. Найти наименьший элемент в таблице значений {ai,bi}.
   2. Запомнить индекс строки таблицы с наименьшим элементом, и добавить его в массив, который хранит оптимальную последовательность
   3. Если этот наименьший элемент принадлежит столбцу значений ai, то деталь с соответствующим номером i поставить на первое свободное место в массив, который соответствует определенному столбцу таблицы,(в первый раз это будет первое место, во второй – второе и т.д.); если же наименьший элемент принадлежит столбцу значений bi, то деталь с соответствующим номером i поставить на последнее свободное место в массив, который соответствует определенному столбцу таблицы (в первый раз это будет n-е место, во второй – (n - 1) и т.д.); если одновременно найдется несколько одинаковых наименьших элементов, то среди них можно выбирать любой, принять за наименьший и поступать так, как описано в начале пункта b алгоритма.
   4. Из таблицы значений {ai,bi} вычеркнуть строку, соответствующую выбранному наименьшему элементу, и проверить, остались ли еще не вычеркнутые строки.
   5. Если не вычеркнутые строки еще остались, то рассматривать их как новую таблицу значений {ai,bi} и перейти к пункту a алгоритма; если вычеркнуты все строки таблицы, то это означает, что алгоритм свою работу закончил.
4. Исходя из полученной последовательности формируем новую таблицу nx3
5. Рассчитываем время обработки
   1. Находим сумму простоев на третьем станке по формуле:
   2. К сумме простоев прибавляем
6. Рисуем график Ганта, используя формулы нахождения простоев на втором и третьем станке:

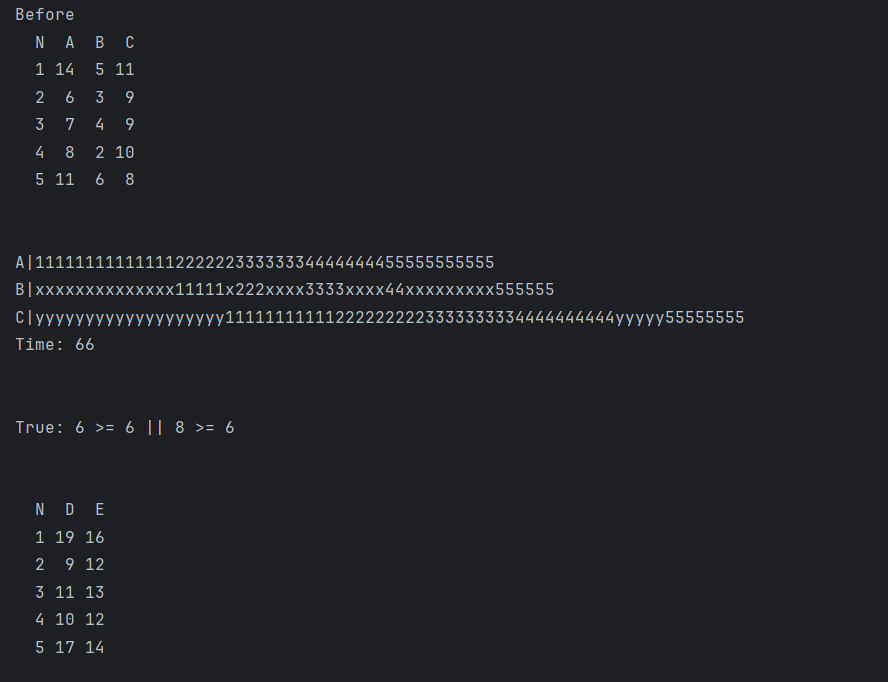
Вариант №23

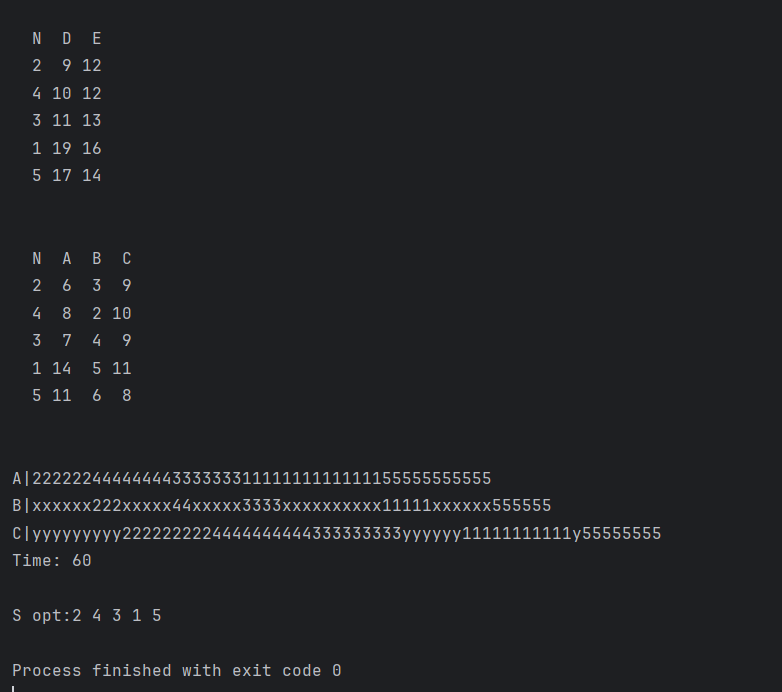
|  |  |  |  |
| --- | --- | --- | --- |
| i | ai | bi | ci |
| 1 | 14 | 5 | 11 |
| 2 | 6 | 3 | 9 |
| 3 | 7 | 4 | 9 |
| 4 | 8 | 2 | 10 |
| 5 | 11 | 6 | 8 |

Решение

public class Main {  
  
 static void drawDiagram(int[] s, int[] a, int[] b, int[] c) {  
 System.*out*.print("A|");  
 for (int i = 0; i < s.length; ++i){  
 for (int j = 0; j < a[ s[i] - 1 ]; ++j)  
 System.*out*.print(s[i]);  
 }  
  
 System.*out*.print("\nB|");  
 for (int i = 0; i < s.length; ++i){  
 int x;  
 if (i == 0)  
 x = a[s[i] - 1];  
 else  
 x = Math.*max*(a[s[i] - 1] - b[s[i - 1] - 1], 0);  
 for (int j = 0; j < x; ++j)  
 System.*out*.print("x");  
 for (int j = 0; j < b[s[i] - 1]; ++j)  
 System.*out*.print(s[i]);  
 }  
  
 System.*out*.print("\nC|");  
 for (int i = 0; i < s.length; ++i){  
 int x;  
 int y;  
 if (i == 0){  
 x = a[ s[i] - 1];  
 y = x + b[ s[i] - 1];  
 }  
 else{  
 x = Math.*max*(a[ s[i] - 1] - b[s[i - 1] - 1], 0);  
 y = Math.*max*(x + b[ s[i] - 1] - c[s[i - 1] - 1], 0);  
 }  
  
 for (int j = 0; j < y; ++j)  
 System.*out*.print("y");  
  
 for (int j = 0; j < c[s[i] - 1]; ++j)  
 System.*out*.print(s[i]);  
 }  
  
 System.*out*.println("\nTime: " + *CalculatTime*(s, a, b, c));  
  
 }  
 static void print(int[] s, int[] a, int[] b, int[] c)  
 {  
 System.*out*.printf("%3c%3c%3c%3c\n", 'N', 'A', 'B', 'C');  
  
 for (int i = 0; i < a.length; ++i)  
 {  
 System.*out*.printf("%3d%3d%3d%3d\n", s[i], a[s[i] - 1 ], b[ s[i] - 1 ], c[ s[i] - 1 ]);  
 }  
 }  
  
 static void print(int[] s, int[] d, int[] e) {  
 System.*out*.printf("%3c%3c%3c\n", 'N', 'D', 'E');  
  
 for(int i = 0; i < d.length; ++i) {  
 System.*out*.printf("%3d%3d%3d\n", s[i], d[i], e[i]);  
 }  
  
 }  
  
 static int CalculatTime(int[] s, int[] a, int[] b, int[] c){  
 int maxKH = Integer.*MIN\_VALUE*;  
 int k = 0;  
 int h = 0;  
 int bPrev = 0;  
 int cPrev = 0;  
 int sumC = 0;  
  
 for (int i = 0; i < s.length; ++i){  
 sumC += c[s[i] -1];  
 k = k + a[s[i] -1] - bPrev;  
 bPrev = b[s[i] -1];  
  
 h = h + b[s[i] -1] - cPrev;  
 cPrev = c[s[i] -1];  
  
 maxKH = ((k + h) > maxKH)? k + h : maxKH;  
 }  
 return sumC + maxKH;  
 }  
  
  
 static void jhonson(int[] s, int[] d, int[] e){  
 int dIndex = 0;  
 int eIndex = d.length;  
  
 // пока для выбора не остается одна строка  
 while (eIndex - dIndex > 1){  
 int minD = e[dIndex];  
 int minE = e[dIndex];  
 int rowD = dIndex;  
 int rowE = dIndex;  
  
 // поиск наименьшего элемента  
 for (int j = dIndex; j < eIndex; ++j) {  
 if (minD > d[j]) {  
 minD = d[j];  
 rowD = j;  
 }  
  
 if (minE > e[j]) {  
 minE = e[j];  
 rowE = j;  
 }  
 }  
  
 if (minD < minE) {  
 int temp = s[dIndex];  
 s[dIndex] = s[rowD];  
 s[rowD] = temp;  
  
 temp = d[dIndex];  
 d[dIndex] = d[rowD];  
 d[rowD] = temp;  
  
 temp = e[dIndex];  
 e[dIndex] = e[rowD];  
 e[rowD] = temp;  
  
 ++dIndex;  
 }  
 else {  
 int temp = s[eIndex - 1];  
 s[eIndex - 1] = s[rowE];  
 s[rowE] = temp;  
  
 temp = d[eIndex - 1];  
 d[eIndex - 1] = d[rowE];  
 d[rowE] = temp;  
  
 temp = e[eIndex - 1];  
 e[eIndex - 1] = e[rowE];  
 e[rowE] = temp;  
  
 --eIndex;  
 }  
  
 }  
 }  
  
 static boolean NextSet(int[] s)  
 {  
 int j = s.length - 2;  
 while (j != -1 && s[j] >= s[j + 1]) j--;  
 if (j == -1)  
 return false; // больше перестановок нет  
  
 int k = s.length - 1;  
 while (s[j] >= s[k])  
 k--;  
  
  
 int temp = s[j];  
 s[j] = s[k];  
 s[k] = temp;  
  
  
 int l = j + 1, r = s.length - 1; // сортируем оставшуюся часть последовательности  
 while (l < r) {  
 l++;  
 r--;  
 temp = s[l];  
 s[l] = s[r];  
 s[r] = temp;  
 }  
 return true;  
 }  
  
 static int max(int[] a){  
 int maxNum = a[0];  
 for (int j : a) {  
 if (j > maxNum)  
 maxNum = j;  
 }  
 return maxNum;  
 }  
  
 static int min(int[] a){  
 int minNum = a[0];  
 for (int j : a) {  
 if (j < minNum)  
 minNum = j;  
 }  
 return minNum;  
 }  
  
  
 public static void main(String[] args) {  
 int[] s = {1, 2, 3, 4, 5};  
 int[] a = {14, 6, 7, 8, 11};  
 int[] b = {5, 3, 4, 2, 6};  
 int[] c = {11, 9, 9, 10, 8};  
  
 System.*out*.println("Before");  
 *print*(s, a, b, c);  
  
 int maxB = *max*(b);  
 int minA = *min*(a);  
 int minC = *min*(c);  
  
  
 System.*out*.println();  
 System.*out*.println();  
 *drawDiagram*(s, a, b, c);  
 System.*out*.println();  
 System.*out*.println();  
  
 if ((minA >= maxB) || (minC >= maxB))  
 {  
 System.*out*.println("True: " + minA + " >= " + maxB + " || " + minC + " >= " + maxB);  
 int[] e = c.clone(), d = a.clone();  
 for (int i = 0; i < d.length; ++i)  
 {  
 d[i] += b[i];  
 e[i] += b[i];  
 }  
  
 System.*out*.println();  
 System.*out*.println();  
 *print*(s, d, e);  
  
 *jhonson*(s, d, e);  
 System.*out*.println();  
 System.*out*.println();  
  
 *print*(s, d, e);  
 } else {  
 System.*out*.println("False: " + minA + " >= " + maxB + " || " + minC + " >= " + maxB);  
  
 int[] sOpt = s;  
 int timeOpt = *CalculatTime*(s, a, b, c);  
 while (*NextSet*(s))  
 {  
 int time = *CalculatTime*(s, a, b ,c);  
 if (time < timeOpt)  
 {  
 timeOpt = time;  
 sOpt = s;  
 }  
 }  
 s = sOpt;  
 }  
  
 System.*out*.println();  
 System.*out*.println();  
  
 *print*(s, a, b, c);  
  
 System.*out*.println();  
 System.*out*.println();  
  
 *drawDiagram*(s, a, b, c);  
 System.*out*.print("\nS opt:");  
 for (int i = 0; i < s.length; ++i)  
 {  
 System.*out*.print(s[i] + " ");  
 }  
 System.*out*.println();  
  
 }  
  
}

Вывод





# **Задание 3. Задача упорядочивания nx3 решаемая методом перебора**

Не выполнение условия Джонсона



не дает возможности свести задачу упорядочения nx3 к задаче упорядочения nx2. Для поиска оптимального решения остается лишь метод перебора, который будет гарантировать, что найденное решение будет самым оптимальным. Общее количество вариантов решения задачи равно n!, где n – количество деталей.

Предположить, что исходный вариант задания 2 не удовлетворяет условию (2.3) и решить задачу методом перебора.

Для создания таблицы исходных данных для задачи Джонсона с тремя станками, необходимо взять соответствующий вариант задания для задачи Джонсона nx3 из методички и откорректировать таблицу так, чтобы не выполнялось условие Джонсона.

**Вариант 23:**

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 14 | 15 | 11 |
| 2 | 6 | 3 | 9 |
| 3 | 7 | 4 | 9 |
| 4 | 8 | 2 | 10 |
| 5 | 11 | 6 | 8 |

# **Решение задания 3**

## Алгоритм решения:

1. Проверяем таблицу nx3 на выполнение условия сводимости к nx2
2. Если условие не выполняется, то методом перестановки
   1. Для реализации выбран рекурсивный метод перестановки. Согласно которому каждый элемент в последовательности переставляется всеми возможными способами. Количество таких способов равно n. Всего возможных последовательностей n!.
   2. После нахождения всех последовательностей, выполняется поиск оптимальной. Для этого рассчитывается время обработки, для каждой таблицы, сформированной в соответствии с найденной последовательностью из массива. Последовательность с минимальным временем обработки является оптимальной.
   3. Время обработки высчитывается по формулам:

.

1. Исходя из полученной последовательности формируем новую таблицу nx3
2. Рисуем график Ганта, используя формулы нахождения простоев на втором и третьем станке:

## Код программы на языке Java

Код программы можно найти в решении задания 2

# **Результаты выполнения задания 3**

