10292. Moortal Cowmbat

Bessie has been playing the popular game Moortal Cowmbat for a long time. However, the game developers have recently released an update that forces her to change her usual play style.

The game uses *m* buttons, labeled with the first *m* lowercase letters of the Latin alphabet. Bessie’s favorite combination of moves is a string *s*  of length *m*, describing a sequence of button presses. After the update, each combo must consist of a sequence of “stripes”, where a stripe is defined as a consecutive sequence of the same button pressed at least *k* times in a row. Bessie wants to modify her favorite combination to obtain a new string of the same length *n*, but consisting of stripes that satisfy the updated rules.

It is known that Bessie needs *aij* days to learn to press button *j* instead of button *i* in any specific position of her combination (that is, replacing one occurrence of letter *i* in *s* with letter *j* costs *aij* days).

Note that sometimes the replacement can be done faster through intermediate buttons. For example, changing *i* directly to *j* may be more expensive than performing two consecutive replacements *i* → *k* → *j*. Thus, there may exist a transformation path from *i* to *j* with a smaller total cost than the direct replacement.

Help Bessie determine the minimum number of days required to create a combination that satisfies the new requirements.

**Input.** The first line contains three integers: *n* (1 ≤ *n* ≤ 105), *m* (1 ≤ *m* ≤ 26) and *k* (1 ≤ *k* ≤ *n*).

The second line contains the string *s*.

The next *m* lines each contain *m* integers – the elements of the matrix *aij*, where *aij* is the number of days required to replace button *i* with button *j*. It is guaranteed that 0 ≤ *aij* ≤ 1000 and *aii* = 0 for all *i*.

**Output.** Print the minimum number of days Bessie needs to create a combination that meets the new requirements.

|  |  |
| --- | --- |
| **Sample input** | **Sample output** |
| 5 5 2abcde0 1 4 4 42 0 4 4 46 5 0 3 25 5 5 0 43 7 0 5 0 | 5 |

## SOLUTION

**dynamic programming**

# Анализ алгоритма

Запустим алгоритм Флойда-Уоршелла на матрице *aij*, чтобы определить кратчайшее время замены каждой буквы на любую другую букву.

Составим матрицу стоимостей *cst* такую что *cst*[*i*][*j*] (1 ≤ *i* ≤ *n*) содержит наименьшее количество дней, за которое можно изменить букву *s*[*i* – 1] на букву *j*.

Для удобства дальнейших вычислений построим матрицу *ps*, содержащую значения частичных сумм матрицы стоимостей *cst* по столбцам:

*ps*[*i*][*j*] = *cst*[*i*][1] + *cst*[*i*][2] + … + *cst*[*i*][*j*]

Пусть *dp*[*i*][*j*] содержит наименьшее количество дней, за которое можно преобразовать первые *i* букв строки *s* в строку, удовлетворяющую новым требованиям Бесси, причем последние *k* букв в новой строке будут буквой *j*.

Запишем уравнения динамики:

* Пусть первая *i* – 1 буква строки *s* уже преобразована в строку, в которой последними *k* буквами стоит буква *j*. Тогда мы просто изменяем букву s[*i* – 1] на *j*, что можно сделать за *cst*[*i*][*j*] дней.

*dp*[*i*][*j*] = *dp*[*i* – 1][*j*] + *cst*[*i*][*j*]

* Чтобы в новой строке последними *k* буквами была буква *j*, рассмотрим значения *dp*[*i* – *k*][0], *dp*[*i* – *k*][1], …, *dp*[*i* – *k*][*m* – 1]. Выберем среди них наименьшее. То есть преобразуем первые *i* – *k* букв строки *s* в строку, удовлетворяющую новым требованиям Бесси (при этом нам не важна буква, которой будет оканчиваться подстрока длины *i* – *k*, нам важно чтобы эту строку получить за наименьшее количество дней). После чего *k* последних букв *s*[*i* – *k*], *s*[*i* – *k* + 1], …, , *s*[*i* – 1] исходной строки *s* меняем на букву *j*, на что уйдет *cst*[*i* – *k* + 1][*j*] + *cst*[*i* – *k* + 2][*j*] + … + *cst*[*i*][*j*] дней. Указанную сумму можно вычислить за константное время, используя массив префиксов *ps*:

*cst*[*i* – *k* + 1][*j*] + *cst*[*i* – *k* + 2][*j*] + … + *cst*[*i*][*j*] = *ps*[*i*][*j*] – *ps*[*i* – *k*][*j*]

Будем поддерживать дополнительный массив *mn*, где

*mn*[*i*] = min(*dp*[*i*][0], *dp*[*i*][1], …, *dp*[*i*][*m* – 1])

Таким образом, уравнение динамики для этого случая (*i* ≥ *k*) будет иметь вид:

*dp*[*i*][*j*] = *ps*[*i*][*j*] – *ps*[*i* – *k*][*j*] + *mn*[*i* – *k*]

Остается среди двух вариантов для *dp*[*i*][*j*] выбрать наименьшее.

Ответом на задачу будет значение

*mn*[*n*] = min(*dp*[*n*][0], *dp*[*n*][1], …, *dp*[*n*][*m* – 1])

Нам не важно, на какие именно *k* букв заканчивается новое слово Бесси.

**Example**

In this example, the optimal solution is as follows: replace *a* with *b*, then *d* with *e*, and finally both *e* with *c*. The total cost of these changes is 1 + 4 + 0 + 0 = 5 days, and the resulting string will be *bbccc*.

**Реализация алгоритма**

Объявим рабочие массивы.

#define MAXN 100005

#define ALPH 26

int d[ALPH][ALPH], cst[MAXN][ALPH], ps[MAXN][ALPH], dp[MAXN][ALPH],

 mn[MAXN];

Читаем входные данные.

cin >> n >> m >> k;

cin >> s;

for (i = 0; i < m; i++)

for (j = 0; j < m; j++)

 cin >> d[i][j];

Запускаем алгоритм Флойда-Уоршелла на матрице *d*. По его завершению d[*i*][*j*] содержит наименьшее количество дней, за которое можно изменить букву *i* на букву *j*.

for (x = 0; x < m; x++)

for (i = 0; i < m; i++)

for (j = 0; j < m; j++)

 d[i][j] = min(d[i][j], d[i][x] + d[x][j]);

У нас имеется *m* ≤ 26 букв. Составим матрицу стоимостей *cst* такую что *cst*[*i*][*j*] (1 ≤ *i* ≤ *n*) содержит наименьшее количество дней, за которое можно изменить букву *s*[*i* – 1] на букву *j*. Индексация букв в строке *s* начинается с 0.

Для удобства дальнейших вычислений построим матрицу *ps*, содержащую значения частичных сумм матрицы стоимостей *cst* по столбцам:

*ps*[*i*][*j*] = *cst*[*i*][1] + *cst*[*i*][2] + … + *cst*[*i*][*j*]

for (i = 1; i <= n; i++)

for (j = 0; j < m; j++)

{

 cst[i][j] = d[s[i - 1] - 'a'][j];

 ps[i][j] = ps[i - 1][j] + cst[i][j];

}

Инициализируем массивы.

memset(dp, 0x3f, sizeof dp);

memset(mn, 0x3f, sizeof mn);

mn[0] = 0;

Вычисляем значения ячеек массива *dp*. Букве *i* соответствует символ *s*[*i* – 1]. Нумерация букв в переменной *j* идет от 0 (которой соответствует буква ‘*a*’) до *m* – 1.

for (i = 1; i <= n; i++)

for (j = 0; j < m; j++)

{

Букву исходной строки *s*[*i* – 1] меняем на *j*.

 dp[i][j] = min(dp[i][j], dp[i - 1][j] + cst[i][j]);

 if (i >= k)

 dp[i][j] = min(dp[i][j], ps[i][j] - ps[i - k][j] + mn[i - k]);

 mn[i] = min(mn[i], dp[i][j]);

}

Выводим ответ.

cout << mn[n] << "\n";