Task: BIR
Birthday

Official English Version

Day 2. Source file `bir.*` *Monday, 22–08–2005*

Available memory: 32 MB. Maximum running time: 2 s.

It is Byteman's birthday today. There are n children at his birthday party (including Byteman). The children are numbered from 1 to n. Byteman's parents have prepared a big round table and they have placed n chairs around the table. When the children arrive, they take seats. The child number 1 takes one of the seats. Then the child number 2 takes the seat on the left. Then the child number 3 takes the next seat on the left, and so on. Finally the child number n takes the last free seat, between the children number 1 and $n-1$.

Byteman's parents know the children very well and they know that some of the children will be noisy, if they sit too close to each other. Therefore the parents are going to reseat the children in a specific order. Such an order can be described by a permutation $p_1, p_2, \ldots, p_n$ ($p_1, p_2, \ldots, p_n$ are distinct integers from 1 to n) — child p_1 should sit between p_n and p_2, child p_i (for $i = 2, 3, \ldots, n-1$) should sit between p_{i-1} and p_{i+1}, and child p_n should sit between p_{n-1} and p_1. Please note, that child p_1 can sit on the left or on the right from child p_n.

To seat all the children in the given order, the parents must move each child around the table to the left or to the right some number of seats. For each child, they must decide how the child will move — that is, they must choose a direction of movement (left or right) and distance (number of seats). On the given signal, all the children stand up at once, move to the proper places and sit down.

The reseating procedure throws the birthday party into a mess. The mess is equal to the largest distance any child moves. The children can be reseated in many ways. The parents choose one with minimum mess. Help them to find such a way to reseat the children.

Task

Your task is to write a program that:

- reads from the standard input the number of the children and the permutation describing the desired order of the children,
- determines the minimum possible mess,
- writes the result to the standard output.

Input

The first line of standard input contains one integer n ($1 \leq n \leq 1\,000\,000$). The second line contains n integers $p_1, p_2, \ldots, p_n$, separated by single spaces. Numbers $p_1, p_2, \ldots, p_n$ form a permutation of the set $\{1, 2, \ldots, n\}$ describing the desired order of the children. Additionally, in 50% of the test cases, n will not exceed 1 000.

Output

The first and the only line of standard output should contain one integer: the minimum possible mess.

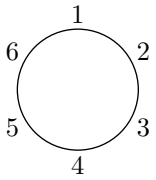
Example

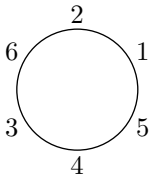
For the input data:

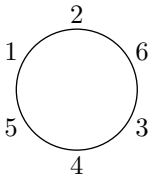
```
6
3 4 5 1 2 6
```

the correct result is:

```
2
```







The left figure shows the initial arrangement of the children. The middle figure shows the result of the following reseating: children number 1 and 2 move one place, children number 3 and 5 move two places, and children number 4 and 6 do not change places. The conditions of arrangement are fulfilled, since 3 sits between 6 and 4, 4 sits between 3 and 5, 5 sits between 4 and 1, 1 sits between 5 and 2, 2 sits between 1 and 6, and 6 sits between 2 and 3. There exists another possible final arrangement of children, depicted in the right figure. In both cases no child moves more than two seats.

Task: REC
Rectangle Game

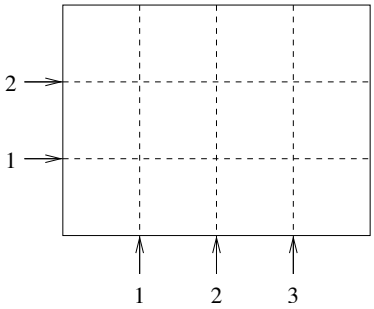
Official English Version

Nowy Sącz, POLAND 2005
The 17-th International
Olympiad in Informatics

Day 2. Source file `rec.*` *Monday, 22–08–2005*

Available memory: 32 MB. Maximum running time: 14 s*.

We consider a two-player game. The players are given an $x \times y$ rectangle (where x and y are positive integers). The players take turns moving. A move consists of dividing a rectangle into two rectangles with a single vertical or horizontal cut. The resulting rectangles must have positive integer dimensions.



Possible cuts of a 4×3 rectangle.

After each cut, the smaller rectangle (that is the one with smaller area) is discarded and the other one is passed to the other player. If the rectangle is cut into two equal halves, then one half is discarded. The player who receives a 1×1 rectangle, and therefore is not able to make a move, loses the game.

Your task is to write a program to play and win the rectangle game. The program must use a special library to play the game. The library provides you with functions `dimension_x()` and `dimension_y()` that return the dimensions of the rectangle. Initial dimensions of the rectangle are integers from 1 to 100 000 000. At least one dimension is greater than 1. Moreover, in 50% of test cases the dimensions do not exceed 25.

There is also a procedure `cut(dir, position)`, that is called by your program to make moves. Parameters `dir` and `position` describe the direction and the position of a cut respectively. The parameter `dir` must be one of the two values: `vertical` or `horizontal`. If `dir` = `vertical` then the cut is vertical, and the parameter `position` specifies the x-coordinate of the cut (see the figure above) and you must ensure that $1 \leq$ `position` $\leq$ `dimension_x`$() - 1$. If `dir` = `horizontal`, then the cut is horizontal and the parameter `position` specifies the y-coordinate of the cut and you must ensure that $1 \leq$ `position` $\leq$ `dimension_y`$() - 1$.

When your program is started, it will act as one player for one game. Your program plays first — it must cut the initial rectangle. When your program calls the `cut` procedure, your move is recorded and control is passed to your program's opponent. After the opponent moves, control returns to your program. Values returned by `dimension_x()` and `dimension_y()` will reflect the result of your move and your opponent's move. As soon as your program wins, loses or makes an illegal move (i.e. calls the `cut` procedure with invalid parameters) it will be terminated. Termination of your program is an automatic process, so your program should keep making moves as long as possible. You can assume that for the test data, there always exists a winning strategy for your program.

Your program must not read or write any files, it must not use standard input/output, and it must not try to modify any memory outside your program. Violating any of these rules may result in disqualification.

*You can assume that during grading library overhead will not exceed 4 s.

Experimentation

To let you experiment with the library, you are given example opponent libraries: their sources are in `preclib.pas`, `creclib.c` and `creclib.h` files. The library can be downloaded from `http://contest/`. They implement a very simple strategy. When you run your program, it will be playing against these simple players. Feel free to modify them, and test your program against a better opponent. However, during the evaluation, your program will be playing against a different opponent.

When you submit your program using the TEST interface it will be compiled with the unmodified example opponent library. The submitted input file will be given to your program standard input. The input file should consist of two lines, each containing one integer. The first line should contain the initial width, and the second line should contain the initial height of the rectangle. These dimensions are read by the example opponent library.

If you modify the implementation part of the `preclib.pas` library, please recompile it using the following command: `ppc386 -O2 preclib.pas`. This command produces files `preclib.o` and `preclib.ppu`. These files are needed to compile your program, and should be placed in the directory, where your program is located. Please do not modify the interface part of the `preclib.pas` library.

If you modify the `creclib.c` library, please remember to place it (together with `creclib.h`) in the directory, where your program is located — they are needed to compile it. Please do not modify the `creclib.h` file.

You are also provided with two simple programs illustrating usage of the above libraries: `crec.c` and `prec.pas`. (Please remember, that these programs are not correct solutions.) You can compile them using the following commands:

```
gcc -O2 -static crec.c creclib.c -lm
g++ -O2 -static crec.c creclib.c -lm
ppc386 -O2 -XS prec.pas
```

Library

You are given a library providing the following functionality:

- **FreePascal Library** (`preclib.ppu`, `preclib.o`)

  ```
  type direction = (vertical, horizontal);
  function dimension_x(): longint;
  function dimension_y(): longint;
  procedure cut(dir: direction; position: longint);
  ```

  Include the following statement in your source file `rec.pas`:
  `uses preclib;`
  To compile your program, copy the files `preclib.o` and `preclib.ppu` to the directory, where your source file is placed and run the following command:
  `ppc386 -O2 -XS rec.pas`

  File `prec.pas` gives an example of how to use the `preclib` library.

- **GNU C/C++ Library** (`creclib.h`, `creclib.c`)

```
typedef enum __direction {vertical, horizontal} direction;
int dimension_x();
int dimension_y();
void cut(direction dir, int position);
```

Include the following statement in your source file (`rec.c` or `rec.cpp`):
`#include "creclib.h"`
To compile your program, copy the files `creclib.c` and `creclib.h` to the directory, where your source file is placed and run the following command:
`gcc -O2 -static rec.c creclib.c -lm`
or:
`g++ -O2 -static rec.cpp creclib.c -lm`
The file `crec.c` gives an example of how to use the library in C.

Sample interaction

Below there is a sample interaction between your program and the judging library. It shows how a sample game can proceed. The game starts with a 4×3 board. There exists a winning strategy for this position.

Your program calls	**What happens**
`dimension_x()`	returns 4
`dimension_y()`	returns 3
`cut(vertical, 1)`	your cut is recorded and a 3×3 board is passed to your opponent, who cuts it to a 3×2 board; after this, control is passed back to your program
`dimension_x()`	returns 3
`dimension_y()`	returns 2
`cut(horizontal, 1)`	your cut is recorded and a 3×1 board is passed to your opponent, who cuts it to a 2×1 board; after this, control is passed back to your program
`dimension_x()`	returns 2
`dimension_y()`	returns 1
`cut(vertical, 1)`	your cut gives a 1×1 board, so you win; your program is terminated automatically

Task: RIV
Rivers

Official English Version

Day 2. Source file `riv.*` *Monday, 22–08–2005*

Available memory: 32 MB. Maximum running time: 1 s.

Nearly all of the Kingdom of Byteland is covered by forests and rivers. Small rivers meet to form bigger rivers, which also meet and, in the end, all the rivers flow together into one big river. The big river meets the sea near Bytetown.

There are n lumberjacks' villages in Byteland, each placed near a river. Currently, there is a big sawmill in Bytetown that processes all trees cut in the Kingdom. The trees float from the villages down the rivers to the sawmill in Bytetown. The king of Byteland decided to build k additional sawmills in villages to reduce the cost of transporting the trees downriver. After building the sawmills, the trees need not float to Bytetown, but can be processed in the first sawmill they encounter downriver. Obviously, the trees cut near a village with a sawmill need not be transported by river. It should be noted that the rivers in Byteland do not fork. Therefore, for each village, there is a unique way downriver from the village to Bytetown.

The king's accountants calculated how many trees are cut by each village per year. You must decide where to build the sawmills to minimize the total cost of transporting the trees per year. River transportation costs one cent per kilometre, per tree.

Task

Write a program that:

- reads from the standard input the number of villages, the number of additional sawmills to be built, the number of trees cut near each village, and descriptions of the rivers,
- calculates the minimal cost of river transportation after building additional sawmills,
- writes the result to the standard output.

Input

The first line of input contains two integers: n — the number of villages other than Bytetown ($2 \leq n \leq 100$), and k — the number of additional sawmills to be built ($1 \leq k \leq 50$ and $k \leq n$). The villages are numbered $1, 2, \ldots, n$, while Bytetown has number 0.

Each of the following n lines contains three integers, separated by single spaces. Line $i+1$ contains:

- w_i — the number of trees cut near village i per year ($0 \leq w_i \leq 10\,000$),
- v_i — the first village (or Bytetown) downriver from village i ($0 \leq v_i \leq n$),
- d_i — the distance (in kilometres) by river from village i to v_i ($1 \leq d_i \leq 10\,000$).

It is guaranteed that the total cost of floating all the trees to the sawmill in Bytetown in one year does not exceed 2 000 000 000 cents.

In 50 % of test cases n will not exceed 20.

Output

The first and only line of the output should contain one integer: the minimal cost of river transportation (in cents).

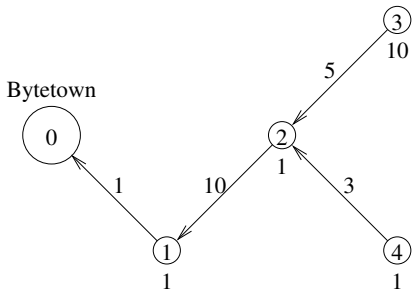
Example

For the input data:

```
4 2
1 0 1
1 1 10
10 2 5
1 2 3
```

the correct result is:

```
4
```



The above picture illustrates the example input data. Village numbers are given inside circles. Numbers below the circles represents the number of trees cut near villages. Numbers above the arrows represent rivers' lengths.

The sawmills should be built in villages 2 and 3.