Алгоритм решения задачи "Пересечение реки"

Антон Дроздовский

1 Условие задачи

На реке имеется n островков, пронумерованных от 1 до n слева направо поперёк реки. Жители должны пересечь реку, начиная с её левого берега и используя некоторые островки для достижения правого берега. Левый берег расположен на одну позицию левее первого островка, а правый берег расположен на одну позицию правее островка с номером n.

В момент времени $t = 0$ житель находится на левом берегу реки и имеет целью добраться до правого берега за минимальное время. В каждый момент времени каждый из островков поднят или опущен и житель стоит на островке или на берегу. Житель может стоять на островке только тогда, когда он поднят. Такой островок называется доступным. В начальный момент времени каждый островок опущен, затем островок поднят a моментов времени, затем опущен b моментов времени, поднят a моментов, опущен b и т. д. Константы a и b задаются отдельно для каждого островка. Например, островок, характеризующийся параметрами $a = 2$ и $b = 3$, опущен в момент времени 0, поднят в моменты времени 1 и 2, опущен в моменты времени 3, 4 и 5 т. д. В момент времени $t + 1$ житель может выбрать островок или берег в пределах пяти ближайших с каждой стороны от его местоположения в момент времени t или остаться на месте (если возможно).

Необходимо вычислить минимальный момент достижения правого берега, если такое достижение возможно.

Input

Первая строка содержит число островков ($5 \leq n \leq 1000$).

Каждая из последующих n строк содержит два натуральных числа a и b ($1 \leq a, b \leq 5$). Числа в $(i+1)$-й строке описывают поведение i-го островка.

Output

Выведите минимальный момент достижения правого берега или сообщение No, если пересечение реки невозможно.

Examples

input.txt	output.txt
10 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	No
10 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1	4

2 Описание алгоритма

Время, когда остров опущен и поднят, образует цикл продолжительностью $a+b$. НОК всех циклов опередеяет наибольший период, за который острова повторяют своё поведение. Если нельзя пересечь реку быстрее, чем за НОК. то реку нельзя пересечь.

Используем поиск в ширину. Уровень дерева — количество времени, вершины — острова. Корень дерева — левый берег реки. Первый уровень содержит левый берег реки и первые 5 островов. Последующие уровни имеют +5 островов каждый. Высота дерева не больше НОК циклов. Уровень не может иметь более $n + 2$ островов. На уровне нет опущенных островов и островов, до которых невозможно дойти за промежуток времени равному уровню.

Алгоритм завершается, когда мы обошли всё дерево или когда дошли до правого берега. Дошли до правого берега — выводим уровень. который обходим.

3 Решение задачи на C++

```
1 #define _CRT_SECURE_NO_WARNINGS
```

```
2
3  #include <algorithm>
4  #include <array>
5  #include <cstdio>
6  #include <ios>
7  #include <iostream>
8  #include <numeric>
9  #include <queue>
10 #include <utility>
11 #include <vector>
12
13 int main() {
14         std::ios_base::sync_with_stdio(false);
15         std::cin.tie(nullptr);
16
17         std::freopen("input.txt", "r", stdin);
18         std::freopen("output.txt", "w", stdout);
19
20         short n;
21         std::cin >> n;
22
23         std::array<int, 1000> a;
24         std::array<int, 1000> b;
25         std::array<int, 1000> timeOfCycle;
26
27         short fullCycle = 1;
28
29         for (short i = 0; i < n; ++i) {
30                 std::cin >> a[i] >> b[i];
31                 timeOfCycle[i] = a[i] + b[i];
32
33                 if (fullCycle < 2520) {
34                         fullCycle = std::lcm(fullCycle,
                               timeOfCycle[i]);
35                 }
36         }
37
38         std::queue<std::pair<short, short>> islandsToVisit;
39         short currTime = 1;
40         std::vector<bool> visited(n, false);
41         std::pair<short, short> island;
42         bool found = false;
43         bool visitedFirstIsland = false;
44
45         islandsToVisit.push({ -1, 0 });
46
47         while (islandsToVisit.empty() == false && found ==
               false && currTime <= fullCycle) {
48                 island = islandsToVisit.front();
49                 islandsToVisit.pop();
```

```
50
51                  if (currTime < island.second + 1) {
52                          ++currTime;
53                          std::fill(visited.begin(), visited.
                                end(), false);
54                          visitedFirstIsland = false;
55                  }
56
57                  for (int i = island.first - 5; i <= island.
                        first + 5; ++i) {
58                          if (i >= 0 && i < n && visited[i] ==
                                 false && island.second %
                                timeOfCycle[i] < a[i] && island.
                                second + 1 < fullCycle) {
59                                  islandsToVisit.push({ i,
                                        island.second + 1 });
60                                  visited[i] = true;
61                          }
62                          else if (i == -1 &&
                                visitedFirstIsland == false &&
                                island.second + 1 < fullCycle) {
63                                  islandsToVisit.push({ -1,
                                        island.second + 1 });
64                                  visitedFirstIsland = true;
65                          }
66                          else if (i == n) {
67                                  found = true;
68
69                                  break;
70                          }
71                  }
72          }
73
74          if (found == true) {
75                  std::cout << currTime << '\n';
76          }
77          else {
78                  std::cout << "No\n";
79          }
80
81          return 0;
82  }
```

4 Оценка временной сложности

1. Ввод данных

Вводятся константы a и b для всех n островов — $O(n)$. Для каждого острова высчитывается НОК цикла и предыдущего вычиселенного НОКа — $O(logN)$, где N — максимальное число, которое используется для вычисления НОКа. В нашем случае $N = 2520$ — НОК первых 10 натуральных чисел. $O(nlogN)$

2. BFS

Алгоритм использует BFS, который имеет временную сложность $O(V+E)$, где V — количество вершин, а E — количество рёбер в графе. В нашем случае граф — дерево, поэтому $E = V - 1$. $O(n \cdot fullCycle)$

3. Вывод результата

$O(1)$

4. Итого

$O(n \cdot fullCycle)$

5 Оценка сложности по памяти

Вектора a, b, timeOfCycle и visited занимают $O(n)$ памяти. Для хранения очереди BFS требуется $O(n \cdot fullCycle)$ памяти. Итого — $O(n \cdot fullCycle)$