Solución

Al iterar sobre todas las salas posibles, podemos formular una lista de espacios libres. Estos son bloques consecutivos de salas vacías (ceros '0'). Considera que $l_1, l_2, \ldots, l_k$ es la distancia entre estos espacios. Por ejemplo, considera la siguiente entrada (input):

10001001001000

En este caso, los espacios vacíos son: 3,2,2,3. (Sería útil tener alguna función que tenga algún tipo de salida que represente la distancia máxima y la distancia mínima encontrada en esta lista.)

Un caso en particular en este problema es cuando hay que colocar a un estudiante en medio de los espacios vacíos. En este caso, queremos ubicarlo en la primera sección vacía al medio:

1**000**1001001000⇒1**0x0**1001001000

Sin embargo, en el escenario que haya un espacio vacío al principio o alfinal de una serie de puestos vacantes, la función debe ajustarse al caso en particular:

1000100100**1000**⇒1000100100**100x**

Hay otro caso en el cual se pueda necesitar poner a los dos alumnos en la misma sección vacante, en dicho caso se necesitaría poner a uno de ellos en ⅓ del camino y al otro en ⅔:

10001**00**1001000⇒10001**xx**1001000

Debido a que es difícil encontrar la mejor combinación para alocar a los alumnos, es mejor probar todas para encontrar el valor de D más grande posible.

C++ (Github para mejor legibilidad)

```
#include <iostream>
#include <fstream>
using namespace std;

// Devuelve el numero mas grande entre dos 1s y su indice
int find_largest_interior_gap(string s, int &gap_start)
{
  int biggest_gap = 0, current_start = -1, N = s.length();
  for (int i = 0; i < N; i++)
          if (s[i] == '1')
          {
          if (current_start != -1 && i - current_start > biggest_gap)
          {
          biggest_gap = i - current_start;
          gap_start = current_start;
          }
          current_start = i;
          }
  return biggest_gap;
}

// Devuelve la distancia más pequeña entre dos 1s
int find_smallest_interior_gap(string s)
{
  int smallest_gap = 1000000000, current_start = -1, N = s.length();
  for (int i = 0; i < N; i++)
          if (s[i] == '1')
          {
          if (current_start != -1 && i - current_start < smallest_gap)
          smallest_gap = i - current_start;
          current_start = i;
          }
  return smallest_gap;
}

// Devuelve el espacio más chico después de agregar a un estudiante al grupo más grande
int try_student_in_largest_gap(string s)
{
```

```
  int gap_start, largest_gap = find_largest_interior_gap(s, gap_start);
  if (largest_gap >= 2)
  {
        s[gap_start + largest_gap / 2] = '1';
        return find_smallest_interior_gap(s);
  }
  return -1; // sin espacio!
}

int main(void)
{
  ifstream fin("socdist1.in");
  int N;
  string s, temp_s;
  fin >> N >> s;
  ofstream fout("socdist1.out");
  int answer = 0;

  // Escenario 1.poner a los dos estudiantes en el espacio interno mas grande
  int gap_start, largest_gap = find_largest_interior_gap(s, gap_start);
  if (largest_gap >= 3)
  {
        temp_s = s;
        temp_s[gap_start + largest_gap / 3] = '1';
        temp_s[gap_start + largest_gap * 2 / 3] = '1';
        answer = max(answer, find_smallest_interior_gap(temp_s));
  }

  // Escenario 2. poner estudiantes en las dos puntas
  if (s[0] == '0' && s[N - 1] == '0')
  {
        temp_s = s;
        temp_s[0] = temp_s[N - 1] = '1';
        answer = max(answer, find_smallest_interior_gap(temp_s));
  }

  // Escenario 3. estudiantes a la izquierda + estudiantes en el grupo interior más grande
  if (s[0] == '0')
  {
        temp_s = s;
```

```
        temp_s[0] = '1';
        answer = max(answer, try_student_in_largest_gap(temp_s));
    }

    // Escenario 4. Estudiantes a la derecha + estudiantes en el grupo interior más grande
    if (s[N - 1] == '0')
    {
        temp_s = s;
        temp_s[N - 1] = '1';
        answer = max(answer, try_student_in_largest_gap(temp_s));
    }

    // Escenario 5. Estudiantes en el espacio interior mas grande.  Hecho dos veces.
    if (largest_gap >= 2)
    {
        temp_s = s;
        temp_s[gap_start + largest_gap / 2] = '1';
        answer = max(answer, try_student_in_largest_gap(temp_s));
    }

    fout << answer << "\n";
    return 0;
}
```

Java (Github para mejor legibilidad)

```
import java.io.*;
import java.util.*;

class socdist1 {
	public static void main(String[] args) throws IOException{

	BufferedReader in = new BufferedReader(new FileReader("socdist1.in"));
	PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter("socdist1.out")));

	int N = Integer.parseInt( in.readLine() );
	String s = in.readLine();

	int currd = 100000000;
	int top1 = 1;
	int top2 = 1;
	int topAdd2 = 1;
	int gapstart = -1;
	for (int i = 0; i < N; i++) {
		boolean curr = s.charAt(i) == '1';
		if (curr) {
		if (gapstart == -1) {
		top1 = Math.max(top1, i);
		topAdd2 = Math.max(topAdd2, i/2);
		gapstart = i;
		}
		else {
		int j = (i-gapstart)/2;
		if (j >= top1) {
			top2 = top1;
			top1 = j;
		}
		else if (j > top2) {
			top2 = j;
		}
		topAdd2 = Math.max(topAdd2, (i-gapstart)/3);
		currd = Math.min(currd, i-gapstart);
		gapstart = i;
		}
		}
	}
	if (gapstart == -1) {
		topAdd2 = Math.max(topAdd2, N-1);
	}
	else {
```

```
            int j = N-gapstart-1;
            if (j >= top1) {
            top2 = top1;
            top1 = j;
            }
            else if (j > top2) {
            top2 = j;
            }
      }
      topAdd2 = Math.max(topAdd2, (N-gapstart-1)/2);
      out.println("" + Math.min(Math.max(Math.min(top1, top2), topAdd2),
currd));
      out.close();
      }
}
```

Python (Github para mejor legibilidad)

```
f = open("socdist1.in", "r")

N = int(f.readline())
s = f.readline()

f.close()

# // Devuelve el numero mas grande entre dos 1s y su indice


def find_largest_interior_gap(s):
      biggest_gap = 0
      current_start = -1
      gap_start = 0
      for i in range(N):
      if s[i] == "1":
            current_start = i
      else:
            if i-current_start > biggest_gap and current_start != -1:
                  biggest_gap = i-current_start
                  gap_start = current_start
      return gap_start, biggest_gap + 1

# Devuelve la distancia más pequeña entre dos 1s
```

```
def find_smallest_interior_gap(s):
      s = s.strip("0")
      s = s[1:-1]
      s = s.split("1")
      smallest_gap = len(min(s)) + 1
      return smallest_gap

# Devuelve el espacio más chico después de agregar a un estudiante al grupo
más grande


def try_student_in_largest_gap(s):
      gap_start, largest_gap = find_largest_interior_gap(s)
      if largest_gap >= 2:
      s = s[:gap_start + largest_gap//2] + "1" + \
            s[gap_start + largest_gap//2 + 1:]
      return find_smallest_interior_gap(s)
      return -1  # sin espacio!


answer = 0

# Escenario 1.poner a los dos estudiantes en el espacio interno mas grande
gap_start, largest_gap = find_largest_interior_gap(s)
if largest_gap >= 3:
      temp_s = s
      temp_s = temp_s[:gap_start + largest_gap // 3] + \
      "1" + temp_s[gap_start + largest_gap // 3 + 1:]
      temp_s = temp_s[:gap_start + largest_gap * 2 // 3] + \
      "1" + temp_s[gap_start + largest_gap * 2 // 3 + 1:]
      answer = max(answer, find_smallest_interior_gap(temp_s))


# Escenario 2. poner estudiantes en las dos puntas
if s[0] == "0" and s[-1] == "0":
      temp_s = s
      temp_s = "1" + temp_s[1:-1] + "1"
      answer = max(answer, find_smallest_interior_gap(temp_s))


# Escenario 3. estudiantes a la izquierda + estudiantes en el grupo interior
más grande
if s[0] == "0":
      temp_s = s
      temp_s = "1" + temp_s[1:]
      answer = max(answer, try_student_in_largest_gap(temp_s))
```

```
# Escenario 4. Estudiantes a la derecha + estudiantes en el grupo interior
más grande
if s[-1] == "0":
      temp_s = s
      temp_s = temp_s[:-1] + "1"
      answer = max(answer, try_student_in_largest_gap(temp_s))


# Escenario 5. Estudiantes en el espacio interior mas grande.  Hecho dos
veces.
if largest_gap >= 2:
      temp_s = s
      temp_s = temp_s[:gap_start + largest_gap // 2] + \
      "1" + temp_s[gap_start + largest_gap // 2 + 1:]
      answer = max(answer, try_student_in_largest_gap(temp_s))


f = open("socdist1.out", "w")

f.write(str(answer))

f.close()
```