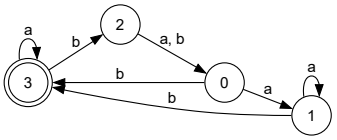
Dopuszczalny czas działania każdego programu wynosi 4 sekundy. Rozmiar pliku źródłowego nie może przekroczyć 32KB. Program nie może wykorzystywać więcej niż 512KB pamięci operacyjnej.

Zad. 1. Deterministyczny automat skończony, w skrócie DAS, to abstrakcyjna maszyna służąca do akceptacji lub odrzucania łańcuchów. Składa się ona ze zbioru stanów ponumerowanych kolejnymi liczbami całkowitymi zaczynając od zera oraz z przejść pomiędzy tymi stanami, które są oznaczone literami *a* i *b*. W danym kroku automat znajduje się dokładnie w jednym stanie i w kolejnym kroku przechodzi do innego stanu (lub pozostaje w tym samym stanie) w zależności od tego, jaką kolejną literę zaobserwuje. Na początku swojej pracy znajduje się on w stanie 0. Część jego stanów (lub wszystkie) są stanami finalnymi. Takie stany finalne na rysunku oznacza się podwójnym okręgiem. Rozważmy przykładowy DAS:



Jeśli będzie on obserwował kolejne litery łańcucha *aabbaa*, to odwiedzi on kolejno stany: 0, 1, 3, 2, 0, 1. Ponieważ nie zatrzymał się w stanie finalnym, łańcuch *aabbaa* zostaje odrzucony (lub inaczej mówiąc, nie jest akceptowany). Natomiast łańcuch *baaa* zostanie zaakceptowany (kolejno odwiedzonymi stanami będą: 0, 3, 3, 3).
Zadanie polega na sprawdzeniu, czy podany automat akceptuje słowa: *aaaa*, *ababab*, *bbb*, *bbbaa* i *aabbbb*.
Dane wejściowe znajdują się w jednym wierszu. Najpierw mamy liczbę n ($1 \leq n \leq 20$) określającą rozmiar automatu (liczbę stanów). Potem mamy n elementowy ciąg zer i jedynek wskazujący, które stany są finalne (0 = nie jest finalny, 1 = jest finalny). Kolejne n liczb określają przejścia z każdego stanu na literę *a*, a ostatnie n liczb określają numery stanów do jakich idziemy na literę *b*.
Na wyjściu ma się znaleźć odpowiedź, pięć liter (t = TAK lub n = NIE), w zależności od tego, czy łańcuchy *aaaa*, *ababab*, *bbb*, *bbbaa* i *aabbbb* są akceptowane przez automat. Poniższy przykład dotyczy automatu pokazanego na wcześniejszym rysunku.
Przykładowe wejście:

```
4 0 0 0 1 1 1 0 3 3 3 0 2
```

Spodziewane wyjście:

```
ntnnt
```

Rozwiązując to zadanie możesz skorzystać z gotowej klasy DFA, którą należy uzupełnić o metodę `accept`.

```
typedef struct Node
{
    bool isFinal = 0;
    struct Node *aInput = nullptr;
    struct Node *bInput = nullptr;
} State;

class DFA
{
public:
    DFA(int n, bool final[], int a[], int b[])
    {
        q = new State[n];
        for (int i = 0; i < n; ++i)
        {
            q[i].isFinal = final[i];
            q[i].aInput = &q[a[i]];
            q[i].bInput = &q[b[i]];
        }
    }

    ~DFA()
    {
        if (q) delete[] q;
        q = nullptr;
    }

    static DFA fromStdInput()
    {
        int i, n;
        bool final[20];
        int a[20];
        int b[20];
        cin >> n;
        for (i = 0; i < n; ++i)
            cin >> final[i];
        for (i = 0; i < n; ++i)
            cin >> a[i];
        for (i = 0; i < n; ++i)
            cin >> b[i];
        return {n, final, a, b};
    }

private:
    State* q = nullptr;
};
```

Zad. 2. Zaokrąglanie to działanie polegające na wyznaczeniu liczby całkowitej, która znajduje się najbliżej zadanej liczby i spełnia określone kryterium. Rozważmy zaokrąglanie liczby do najbliższej liczby Fibonacciego.
Ciąg liczb Fibonacciego zdefiniowany jest następująco:

- $F(0) = 0$,
- $F(1) = 1$,
- $F(n) = F(n-1) + F(n-2)$,

co daje ciąg: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55$ itd.
Zadanie polega więc na tym, aby dla zadanej liczby $x > 0$ znaleźć taką liczbę $F(k)$, żeby $|x - F(k-1)| > |x - F(k)|$ oraz $|x - F(k+1)| \geq |x - F(k)|$. Zadana wartość $x \leq 10^9$ będzie typu `uint32_t`.
Przykładowe wejście:

```
25
```

Spodziewane wyjście:

```
21
```

Zad. 3. Dany jest zbiór $Z = \{1, 2, \ldots, n\}$ ($n \leq 30$) oraz dodatnia liczba całkowita $k \leq \frac{n(1+n)}{2}$. Ile jest podzbiorów zbioru Z, w których suma elementów wynosi k?
Na wejściu dostajemy n i k, a na wyjściu podajemy odpowiednią liczbę.
Przykładowe wejście:

```
4 6
```

Spodziewane wyjście:

```
2
```