Dappy

BWINF Aufgabe 3

Team ID: 00871

Einleitung

Wir werden dieses Problem lösen indem wir diese 3 Dimensionale Matrix (Stockwerk wird hier, obwohl es binär ist, auch als Dimension gezählt) in einen Graph übersetzen. Danach rennen wir, ausgehend von der Startposition, Dijkstra, bis wir unsere Endposition erreicht haben und somit den kürzesten Weg gefunden haben. Um weiter den gefunden Pfad zurückzuverfolgen, benutzen wir ein weiteres Array, zusätzlich zum literarisch-klassischen Distanz Array, für den Dijkstra Algorithmus. Dabei speichern wir unter $\mathrm{pre}[n]$ ab, welchen Knoten wir vorher benutzt haben, um auf diese kürzeste Distanz bei Knoten n anzukommen. Wenn wir also fertig mit Dijkstra sind, müssen wir nur noch diesem Array rückwärts folgen und die Symbole an den richtigen Positionen passend ändern.

Mapping

Um die Matrix in einen Graphen übersetzen und später aber auch wieder zurückzufinden nachdem wir mit dem Dijkstra Algorithmus fertig sind, brauchen wir eine Bijektion zwischen Position $\mathrm{Matrix}[i][j][k]$ und Knoten l . Dabei stellt i das Stockwerk, j die Reihe und k die Spalte in der Matrix da. Es sollte angemerkt werden, dass ich davon ausgegangen bin, dass die ersten und letzten Reihen und Spalten nur aus Wänden bestehen. Dies schränkt nämlich den Algorithmus in keiner Art ein (man kann einfach bei einem Input der dies nicht erfüllt, 2 Extra Spalten und Reihen voller Wände hinzufügen) und war vor allem dadurch motiviert, dass man dann keine weiteren 100 if's braucht um einen Index Error vorzubeugen, wenn man überprüft ob die anliegenden Zellen Wände sind. Schließlich habe ich mich für folgende Bijektion entschieden:

```
#####
#012#
#345#
#####

#####
#678#
#9..#
#####
```

Dabei wird von links nach rechts erst aufgezählt, danach oben nach unten, und final Stockwerk 1 zu Stockwerk 2.

Adjacency List

Jetzt wo wir ein Mapping haben, müssen wir die Verbindungen der Matrix in den Graphen übersetzen. Wir gehen also durch die Matrix, und inkrementieren einen Counter für jede Position die wir besichtigen: Dies ist unsere Bijektion für die jetzige Position in der Matrix. Wir gehen alle 5 Richtungen (oben, unten, links, rechts, Stockwerk wechseln) durch und überprüfen ob dort eine Wand ist oder nicht. Falls nicht, fügen wir es einer Adjacency Liste für den jetzigen Knoten hinzu.

Pseudocode zum erstellen der Adjacency List:

```
bijektion := 0 // stellt den Wert der Bijektion dar

for i von 1 bis Hoehe - 1:
        for j von 1 bis Laenge - 1:
                bijektion++
                falls links von Matrix[i][j][0] keine Wand ist:
                        adjacencyList[bijection] append bijection - 1 // bijektion - 1 stellt fuer die
bijektion das Feld links vom jetztigen Feld dar
                falls rechts von Matrix[i][j][0] keine Wand ist:
                        adjacencyList[bijection] append bijection + 1
```

```
                // wiederhole fuer oben und unten und Stockwerk wechseln

// wiederhole fuer das zweite Stockwerk
```

Start- & Endposition

Um die Start und Endposition zufinden, gehen wir einfach einmal komplett durch unsere Matrix und überprüfen ob dort ein 'A' oder 'B' vielleicht steht. Ist dies der Fall, speichern wir uns die Position, nachdem wir unsere Bijektion auf die Matrixposition angewendet haben, ab.

Dijkstra

Nun haben wir einen Graph und vollständige Adjacency List und Start und Endposition. Wir rennen also einfach normalen Dijkstra darauf. Wenn wir jedoch eine Distanz updaten, speichern wir auch bei welchem Knoten wir vorher waren. Dies brauchen wir, wie früher angemerkt, fürs backtracken. Außerdem könnte man aus effizienz Gründen ein if einbauen dass überprüft ob wir bei der Endposition angelangt sind; ist dies nämlich der Fall, haben wir schon den kürzesten Weg und könnten im Prinzip abbrechen; notwendig ist dies jedoch nicht.

Backtracking

Wir fangen bei der Endposition an und wenden $\mathrm{pre}[n]$ an. Dabei merken wir uns immer die letzten beiden Positionen, damit wir die Richtung auch bestimmen können. Nun wenden wir die Bijektion, rückwärts an, und kriegen wieder die Position des Matrix Eintrags. Mittels ein paar if's können wir auch feststellen, in welche Richtung wir gegangen sind und schließlich an der richtigen Position das Symbol einfügen. Diesen Prozess wiederholen wir einfach bis wir bei der Startposition angelangt sind.

Pseudocode zum Backtracking:

```
curNode := Endposition
previousNode := None

while curNode != Anfangsposition:
        curNode = pre[curNode]
        if(previousNode + 1 = curNode):
                // wir sind nach rechts gegangen, da wir es rueckwaerts verfolgen sind wir eigentlich nach
links gegangen
                Setze Matrix an der entsprechenden Position von curNode auf "<"
        if(previousNode - 1 = curNode):
                Setze Matrix an der entsprechenden Position von curNode auf ">"
        // wiederhole fuer oben unten und Stockwerk wechseln
        previousNode = curNode
```

Ausgabe

Somit sind wir final fertig und geben einfach die Matrix wieder aus. Es sollte angemerkt werden, dass man A im Ergebnis nicht wiederfindet, schließlich wurde es durch ein Symbol ersetzt welches die Richtung andeutet. Mindestens ist B aber noch da, da wenn wir B erreichen automatisch fertig sind und keine Richtung mehr andeuten müssen.

Analyse

Lass n, m die Dimension der Matrix eines Stockwerkes sein. Finden der Start- und Endposition sowohl erstellen der Adjacency List und Backtracking sind alles lineare Option in Respekt zu einer der Variablen: Es gilt $O(nm)$ für den Speicher und Zeit. Dijkstra rennt in $O(V \cdot \log(E))$ (dies kommt sehr auf die Implementation an, zur Einfachheit und da es sich um eine oberflächliche Analyse handelt, wird dies ignoriert), wobei V die Anzahl von Knoten betitelt und E die Kanten. Da Grundsätzlich für jeden Knoten gilt, dass er maximal in alle Richtung eine Kante hat, gilt $E \leq 5 \cdot V$ und es gibt

insgesamt $n \cdot m \cdot 2 = E$ Knoten. Somit ergibt sich $O(nm + nm \cdot \log(nm)) = O(nm \cdot \log(nm))$ oder für $n = m$, $O(n^2 \cdot log(n))$. Folgend sind die Beispieleingaben mit $n, m < 300$ in weniger als einer Sekunde einfach bearbeitbar.

Beispieleingaben

Zauberschule0.txt

Eingabe

```
13 13
#############
#.....#.....#
#.###.#.###.#
#...#.#...#.#
###.#.###.#.#
#...#.....#.#
#.#########.#
#.....#.....#
#####.#.###.#
#....A#B..#.#
#.#########.#
#...........#
#############

#############
#.......#...#
#...#.#.#...#
#...#.#.....#
#.###.#.#.###
#.....#.#...#
#####.###...#
#.....#.....#
#.#########.#
#...#.......#
#.#.#.#.###.#
#.#...#...#.#
#############
```

Ausgabe

```
#############
#.....#.....#
#.###.#.###.#
#...#.#...#.#
###.#.###.#.#
#...#.....#.#
#.#########.#
#.....#.....#
#####.#.###.#
#....!#B..#.#
#.#########.#
#...........#
#############

#############
```

```
#.......#...#
#...#.#.#...#
#...#.#.....#
#.###.#.#.###
#.....#.#...#
#####.###...#
#.....#.....#
#.#########.#
#...#>>!....#
#.#.#.#.###.#
#.#...#...#.#
#############
```

Zauberschule1.txt

Eingabe

```
11 21
#####################
#...#.....#...#.....#
#.#.#.###.#.#.#.###.#
#.#.#...#.#.#...#...#
###.###.#.#.#####.###
#.#.#...#.#B....#...#
#.#.#.###.#.###.#####
#.#...#.#.#..A#.....#
#.#####.#.#########.#
#...................#
#####################

#####################
#.......#.....#.....#
#.###.#.#.###.#.###.#
#.....#.#.#.....#.#.#
#######.#.#######.#.#
#.....#.#.....#...#.#
#.###.#.#.###.###.#.#
#.#.#...#.#...#...#.#
#.#.#######.###.###.#
#...........#.......#
#####################
```

Ausgabe

```
#####################
#...#.....#...#.....#
#.#.#.###.#.#.#.###.#
#.#.#...#.#.#...#...#
###.###.#.#.#####.###
#.#.#...#.#B....#...#
#.#.#.###.#^###.#####
#.#...#.#.#^<<#.....#
#.#####.#.#########.#
#...................#
#####################
```

```
####################
#.......#.....#.....#
#.###.#.#.###.#.###.#
#.....#.#.#.....#.#.#
#######.#.#######.#.#
#.....#.#.....#...#.#
#.###.#.#.###.###.#.#
#.#.#...#.#...#...#.#
#.#.#######.###.###.#
#..........#.......#
####################
```

Zauberschule2.txt

Eingabe

```
15 45
#############################################
#...#.....#..........#.#..........#.......#
#.#.#.###.#########.#.#.#.#######.#.#.#####.#
#.#.#...#.#.........#A..#.#.....#.#.#...#...#
###.###.#.#.##############.#.#.###.#.###.#.###
#.#.#...#.#................B#.#...#.#...#.#.#
#.#.#.###.###########.#########.###.#.###.#.#
#.#...#.#.#.........#.#.#.....#.#.....#.#.#.#
#.#####.#.#.#######.#.#.#.###.#.#######.#.#.#
#.....#...#...#.#...#...#.#.#.#.......#.#.#.#
#.#####.#####.#.#.#######.#.#.#######.#.#.#.#
#.....#.......#.#.#.....#.#.#.#...#...#...#.#
#.###.#########.#.###.#.#.#.#.#.#.#.###.###.#
#...#.................#...#.....#...#.......#
#############################################

#############################################
#...#.......#.....#.....#...#...#.....#.....#
#.#.#.#####.###.#.###.#.#.#.#.#.#.###.###.###
#.#.#.....#.#...#.....#...#.#.#...#.#...#...#
###.#.###.#.#.##############.#.#####.###.###.#
#.#.#...#.#.#.#.....#...#.#.#...#.#...#.#...#
#.#.#####.#.#.#.###.#.#.#.#.#.#.#.#.#.#.#.###
#.#...#...#.#.....#.#.#.#...#.#.#.#...#.#.#...#
#.###.#.###.#.#####.#.###.#.###.#####.#.###.#
#...#.#.#...#...#...#...#.#...#.#.....#.....#
#.###.#.#.#######.#####.#####.#.#.#.#######.#
#...#...#...#...#.....#.......#.#.#.#.....#.#
#.#.#######.#.#.#####.#.#######.#.###.###.#.#
#.#..........#.......#.........#.......#...#
#############################################
```

Ausgabe

```
#############################################
#...#.....#..........#.#..........#.......#
#.#.#.###.#########.#.#.#.#######.#.#.#####.#
```

```
#.#.#...#.#.........#>>!#v#.....#.#.#...#...#
###.###.#.#.##############v#.#.###.#.###.#.###
#.#.#...#.#..............>>B#.#...#.#...#.#.#
#.#.#.###.############.#########.###.#.###.#.#
#.#...#.#.#.........#.#.#.....#.#.....#.#.#.#
#.#####.#.#.########.#.#.#.###.#.#######.#.#.#
#.....#...#...#.#...#...#.#.#.#.......#.#.#.#
#.#####.#####.#.#.########.#.#.########.#.#.#.#
#.....#.......#.#.#.....#.#.#.#...#...#...#.#
#.###.#########.#.###.#.#.#.#.#.#.#.###.###.#
#...#.................#...#.....#...#.......#
###############################################

###############################################
#...#.......#.....#.....#...#...#.....#.....#
#.#.#.#####.###.#.###.#.#.#.#.#.#.###.###.###
#.#.#.....#.#...#.....#>>!#.#.#...#.#...#...#
###.#.###.#.#.##############.#.#####.###.###.#
#.#.#...#.#.#.#.....#...#.#.#...#.#...#.#...#
#.#.#####.#.#.#.###.#.#.#.#.#.#.#.#.#.#.#.###
#.#...#...#.#.....#.#.#...#.#.#.#...#.#.#...#
#.###.#.###.#.#####.#.###.#.###.#####.#.###.#
#...#.#.#...#...#...#...#.#...#.#.....#.....#
#.###.#.#.#######.#####.#####.#.#.#.#######.#
#...#...#...#...#.....#.......#.#.#.#.....#.#
#.#.#######.#.#.#####.#.#######.#.###.###.#.#
#.#...........#.......#.........#.......#...#
###############################################
```

Zauberschule3.txt

Eingabe

```
31 31
################################
#...#.....#...........#.......#
#.#.#.###.#.###.#######.###.#.#
#.#.#...#.#.#.#.#.......#...#.#
###.###.#.#.#.#.#.#######.###.#
#.#.#...#.#...#.....#.....#.#.#
#.#.#.###.###########.#####.#.#
#.#...#.#.#.........#...#.....#
#.#####.#.#.#######.###.#.#####
#...#.#...#...#.#...#...#.#...#
#.#.#.#.#####.#.#.###.###.#.#.#
#.#.#.#.......#.#.#...#.#...#.#
#.#.#.#########.#.#.###.#####.#
#.#.......#.#.....#.#.#.....#.#
#.#######.#.#.#####.#.#.###.#.#
#.#.....#...#.#.#...#.#.#.#...#
#.#.###.#####.#.#.#.#.#.#.#####
#.#...#.....#.#.#.#.#.#...#.....#
#.###.#####.#.#.#.#.###.#####.#
#...#.#...#...#...#...#.......#
#.#.#.#.##############.#.#######
```

```
#.#.#.#.............#.#.....#
#.###.#.#######.#########.###.#
#...#.#.#...#...#.........#...#
###.#.###.#.#.###.#####.###.#.#
#...#.....#.#.....#..B#.#...#.#
#.#########.#######.#.###.###.#
#..A#.....#.#.......#...#.#.#.#
#.#.#.###.#.#.#########.#.#.#.#
#.#.....#.....#...........#...#
################################

################################
#.......#.....#...#...#.......#
#.###.#.#.#.#.###.#.#.#.#####
#.....#.#.#.#.....#.#.#.....#
#######.#.#.#.#.#####.#.#####.#
#.....#.#.#.#.#.#...#...#.....#
#.###.#.###.#.###.#.#.###.#####
#...#.#.#...#.....#.#.#.#.....#
#.#.###.#.#########.#.#.#####.#
#.#.....#.#.......#.#.....#...#
#.#######.#######.#.#.#####.#.#
#...#...#.....#...#.#.#...#.#.#
###.#.#.#.###.#.###.#.#.#.#.#.#
#.#.#.#...#...#.....#.#.#.#.#.#
#.#.#.#####.###.#######.#.#.#.#
#.#.#.....#.#.....#.....#.#.#.#
#.#.#######.#####.###.###.#.#.#
#.#.....#...#...#.....#.#...#.#
#.#####.#.###.#.#######.#####.#
#.#...#.#.#...#.......#.#.....#
#.#.#.#.#.#.#########.#.#.#####
#...#.#.#.#...#...#.#.#.#.....#
#####.#.#.###.#.#.#.#.#.#####.#
#.....#.#.......#.#.#.#...#...#
#.###.#.#########.#.#.#.#.#.###
#...#.#.....#.#.....#.#.#.#.#.#
#.#.#######.#.#.#####.#.#.#.#.#
#.#.......#.#...#.....#.#...#.#
#.###.#####.#####.#####.#####.#
#...#.............#..........#
################################
```

Ausgabe

```
################################
#...#.....#..........#.......#
#.#.#.###.#.###.########.###.#.#
#.#.#...#.#.#.#.#.......#...#.#
###.###.#.#.#.#.#.########.###.#
#.#.#...#.#...#.....#.....#.#.#
#.#.#.###.############.#####.#.#
#.#...#.#.#.........#...#.....#
#.#####.#.#.########.###.#.#####
#...#.#...#...#.#...#...#.#.#...#
```

```
#.#.#.#.#####.#.#.###.###.#.#.#
#.#.#.#.......#.#.#...#.#...#.#
#.#.#.#########.#.#.###.#####.#
#.#.......#.#.....#.#.#.....#.#
#.#######.#.#.#####.#.#.###.#.#
#.#.....#...#.#.#...#.#.#.#...#
#.#.###.#####.#.#.#.#.#.#.#####
#.#...#.....#.#.#.#.#...#.....#
#.###.#####.#.#.#.#.###.#####.#
#...#.#...#...#...#...#.......#
#.#.#.#.##############.#.#######
#.#.#.#..............#.#.....#
#.###.#.#######.#########.###.#
#...#.#.#...#...#.........#...#
###.#.###.#.#.###.#####.###.#.#
#...#.....#.#.....#>>B#.#...#.#
#.#########.#######^#.###.###.#
#..v#>>>>v#.#>>>>>>^#...#.#.#.#
#.#v#^###v#.#^#########.#.#.#.#
#.#>>^..#>>>>^#..........#...#
################################

################################
#.......#.....#...#...#.......#
#.###.#.#.#.#.#.###.#.#.#.#####
#.....#.#.#.#.#.....#.#.#.....#
#######.#.#.#.#.#####.#.#####.#
#.....#.#.#.#.#.#...#...#.....#
#.###.#.###.#.###.#.#.###.#####
#...#.#.#...#.....#.#.#.#.....#
#.#.###.#.#########.#.#.#####.#
#.#.....#.#.......#.#.....#...#
#.#######.#######.#.#.#####.#.#
#...#...#.....#...#.#.#...#.#.#
###.#.#.#.###.#.###.#.#.#.#.#.#
#.#.#.#...#...#.....#.#.#.#.#.#
#.#.#.#####.###.#######.#.#.#.#
#.#.#.....#.#.....#.....#.#.#.#
#.#.#######.#####.###.###.#.#.#
#.#.....#...#...#.....#.#...#.#
#.#####.#.###.#.#######.#####.#
#.#...#.#.#...#.......#.#.....#
#.#.#.#.#.#.#########.#.#.#####
#...#.#.#.#...#...#.#.#.#.....#
#####.#.#.###.#.#.#.#.#.#####.#
#.....#.#.......#.#.#.#...#...#
#.###.#.#########.#.#.#.#.#.###
#...#.#.....#.#.....#.#.#.#.#.#
#.#.#######.#.#.#####.#.#.#.#.#
#.#.......#.#...#.....#.#...#.#
#.###.#####.#####.#####.#####.#
#...#.............#..........#
################################
```

Zauberschule4.txt & Zauberschule5.txt

Lassen sich in separaten .txt files finden, da die Ausgabe und Eingabe zu groß sind um sie in die Dokumentation zu packen

Quellcode

Zuerst befindet sich ein competitive programming spezifisches Template, welches ich auch benutzt habe, jedoch im Prinzip ignoriert werden kann.

```
#include <bits/stdc++.h>

using namespace std;

// Types
typedef long long ll;
typedef string str;

typedef vector<ll> vll;
template <class T> using V = vector<T>;

typedef pair<ll, ll> pll;

typedef priority_queue<pll, V<pll>, greater<pll>> pqll;

// Firsts
#define FAST cin.tie(0); cout.tie(0); ios::sync_with_stdio(0)
#define READ_FILE(i, o) freopen(i, "r", stdin); freopen(o, "w", stdout)
#define PRECISION(n) cout << fixed << setprecision(n)

// Pair
#define MP make_pair

// For
#define rep(i,n) for (ll i = 0; i < (n); i++)
#define each(e,v) for(auto &e : v)

// Input
#define gets(val) cin >> val
template <class T> void get(T &t) {
    cin >> t;
}
template <class T, class... U> void get(T &t, U &... u) {
    get(t); get(u...);
}
template <class T> void get(V<T> &v) {
    each(e, v) get(e);
}

// Output
#define puts(val) cout << val << "\n"
template <class T> void put(T &t) {
    cout << t;
}
template <class T, class... U> void put(T &t, U &... u) {
    put(t); put(u...);
}

#define nxt() cout << "\n"
#define sp() cout << " "

#define PB push_back
#define P push
#define I insert
```

```
const double eps = 1e-9;
const ll INF = 1e16;

void solve() {
    ll n,m; get(n,m);

    V<str> map1(0);

    V<str> map2(0);

    string a;
    rep(i,n){
        get(a);
        map1.PB(a);
    }
    rep(i,n){
        get(a);
        map2.PB(a);
    }

    // find coordinates of start and ending point

    ll startingPoint, endingPoint;

    rep(i,n){
        rep(j,m){
            if(map1[i][j] == 'A'){
                startingPoint = (m-2)*(i-1) + (j-1);
            }else if(map1[i][j] == 'B'){
                endingPoint = (m-2)*(i-1) + (j-1);
            }
        }
    }

    rep(i,n){
        rep(j,m){
            if(map2[i][j] == 'A'){
                startingPoint = (m-2)*(i-1) + (j-1) + (n-2)*(m-2);
            }else if(map2[i][j] == 'B'){
                endingPoint = (m-2)*(i-1) + (j-1) + (n-2)*(m-2);
            }
        }
    }

    V<V<pll>> adj(0);

    ll cur = 0;

    n -= 2;
    m -= 2;

    V<pll> tempAdj = {};

    // build all connections

    // map1 -> map1/map2
    for(int i = 1; i<n+1; i++){
        for(int j = 1; j<m+1; j++){
            tempAdj = {};
            // oben
            if(map1[i-1][j] != '#' && cur > m){
                tempAdj.PB(MP(cur-m, 1));
```

```
        }
        // unten
        if(map1[i+1][j] != '#' && cur + m < n*m){
            tempAdj.PB(MP(cur+m, 1));
        }
        // links
        if(map1[i][j-1] != '#' && cur%m != 1){
            tempAdj.PB(MP(cur-1, 1));
        }
        // rechts
        if(map1[i][j+1] != '#' && cur%m != 0){
            tempAdj.PB(MP(cur+1, 1));
        }
        // stockwerk
        if(map2[i][j] != '#'){
            tempAdj.PB(MP(cur+n*m,3));
        }
        adj.PB(tempAdj);
        cur += 1;
    }
}


// map2 -> map2/map1
for(int i = 1; i<n+1; i++){
    for(int j = 1; j<m+1; j++){
        tempAdj = {};
        // oben
        if(map2[i-1][j] != '#' && cur > m){
            tempAdj.PB(MP(cur-m, 1));
        }
        // unten
        if(map2[i+1][j] != '#' && cur + m < 2*n*m){
            tempAdj.PB(MP(cur+m, 1));
        }
        // links
        if(map2[i][j-1] != '#' && cur%m != 1){
            tempAdj.PB(MP(cur-1, 1));
        }
        // rechts
        if(map2[i][j+1] != '#' && cur%m != 0){
            tempAdj.PB(MP(cur+1, 1));
        }
        // stockwerk
        if(map1[i][j] != '#'){
            tempAdj.PB(MP(cur-n*m,3));
        }
        adj.PB(tempAdj);
        cur += 1;
    }
}

// now just run plain dijikstra on it


pqll pq;

pq.P(MP(0,startingPoint));

vll dist(n*m*2, INF);
vll pre(n*m*2, -1);

ll node, distance;
```

```
    while(!pq.empty()){
        tie(distance, node) = pq.top();
        pq.pop();

        each(e, adj[node]){
            if(distance+e.second < dist[e.first]){
                dist[e.first] = distance+e.second;
                pre[e.first] = node;
                pq.P(MP(distance+e.second, e.first));
            }
        }
    }

    // puts(dist[endingPoint]);

    // backtrack the nodes

    ll backtracking = endingPoint;
    ll temp = endingPoint;
    while(backtracking != startingPoint){
        temp = backtracking;
        backtracking = pre[backtracking];

        // find out where the new index is
        ll row = backtracking % m;
        ll column = (backtracking - row)/m;
        // find out which direction we are going (invert it)
        char sign = '0';
        if(temp + 1 == backtracking){
            sign = '<';
        }else if(temp - 1 == backtracking){
            sign = '>';
        }else if(temp + m == backtracking){
            sign = '^';
        }else if(temp - m == backtracking){
            sign = 'v';
        }else{
            sign = '!';
        }
        if(backtracking < n*m){
            // map 1
            map1[column+1][row+1] = sign;
        }else{
            // map 2
            map2[column-n+1][row+1] = sign;
        }
    }

    // output the result

    puts("OUTPUT:");

    rep(i,n+2){
        put(map1[i]);
        nxt();
    }
    nxt();
    rep(i,n+2){
        put(map2[i]);
        nxt();
    }
}
```

```
int main() {
    READ_FILE("aufgabe3Input.txt", "aufgabe3Output.txt"); // comment this line if input should be done
using console

    solve();

    return 0;
}
```