LAPORAN TUGAS KECIL 1

IF2211 STRATEGI ALGORITMA

PENYELESAIAN CYBERPUNK 2077 BREACH PROTOCOL DENGAN ALGORITMA BRUTE FORCE

Disusun oleh:

Mesach Harmasendro 13522117

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2024

DAFTAR ISI

BAB 1
ALGORITMA

Cara termudah untuk menyelesaikan permasalahan Cyberpunk 2077 Breach Protocol ini adalah dengan cara mencari semua kemungkinan buffer dari panjang yang paling pendek hingga panjang yang paling panjang. Pada setiap akhir proses pencarian akan dihitung reward untuk suatu buffer tersebut, kemudian membandingkannya dengan data yang sudah ada sebelumnya dan akan menyimpan data buffer dengan nilai reward terbesar. Proses pencarian semua kemungkinan tersebut bisa diterapkan ke dalam sebuah program dengan menggunakan algoritma bruteforce dan akan menggunakan pendekatan rekursif untuk lebih mempermudah proses tracking.

Berikut adalah langkah-langkah algoritma bruteforce yang digunakan:

1. Inisiasi semua variabel yang diperlukan seperti variabel sementara untuk menyimpan *max_reward* dan *optimal_buffer*. Proses pencarian atau penelusuran akan dibantu dengan menggunakan fungsi rekursif yang menerima beberapa argumen yaitu *buff_length, current_buff, is_vertical, baris,* dan *kolom.* Parameter *buff_length* digunakan untuk menyimpan informasi mengenai sisa token yang diperlukan hingga buffer mencapai panjang yang diinginkan. Parameter *current_buff* digunakan untuk menyimpan sementara data buffer saat ini. Parameter *is_vertical* digunakan untuk menentukan arah penelusuran saat ini apakah penelusuran secara vertical atau horizontal. Parameter *baris* dan *kolom* digunakan untuk menyimpan posisi penelusuran saat ini dalam sebuah matriks.
2. Karena proses pencarian akan selalu dimulai pada baris pertama maka proses pencarian pertama akan dimulai dari baris pertama kolom pertama. Proses penelusuran kemudian akan berlangsung dimulai dari penelusuran secara vertikal kemudian horizontal kemudian vertikal lagi dan begitu seterusnya (penelusuran secara vertikal dan horizontal akan dilakukan secara bergantian). Setiap token dalam matriks yang dikunjungi akan disimpan sementara ke dalam parameter *current_buff*. Setiap bertambahnya data pada *current_buff* maka nilai dari *buff_length* juga akan berkurang satu. Ketika suatu buffer sudah

mencapai panjang buffer yang diinginkan (*buff_length == 0,* basis rekurens) maka akan dilakukan perhitungan reward (berdasarkan data sequence yang diberikan) pada buffer tersebut dan akan dilakukan perbandingan dengan data yang sudah disimpan sebelumnya pada variabel *max_reward* dan *optimal_buffer.* Jika nilai dari *max_reward* dan *optimal_buffer* masih belum ada/tidak terdefinisi maka nilai dari buffer dan reward saat ini akan menjadi nilai sementara untuk variabel *max_reward* dan *optimal_buffer*. Jika nilai reawrd saat ini lebih besar dari nilai *max_reward* maka data buffer dan reward saat inilah yang akan disimpan di variabel *max_reward* dan *optimal_buffer*. Jika nilai *max_reward* yang lebih besar maka data yang disimpan di variabel *max_reward* dan *optimal_buffer* tidak berubah. Jika nilai *max_reward* sama dengan nilai dari reward saat ini maka akan dilakukan perbandingan panjang buffer dan nilai panjang yang paling pendek lah yang akan disimpan.

3. Setelah fungsi rekurens mencapai basis maka fungsi akan melakukan backtracking dan akan mencari kemungkinan jalur lainnya sesuai dengan paramater pada fungsi rekurens saat itu. Sebelum melanjutkan penelusuran ke lokasi berikutnya akan selalu dilakukan pengecekan terlebih dahulu apakah lokasi yang akan dituju sudah pernah dikunjungi atau tidak (pengecekan dilakukan dengan melihat lokasi dari setiap token yang ada pada buffer saat ini). Jika lokasi sudah pernah dikunjungi sebelumnya maka fungsi akan mencari kemungkinan jalur lainnya. Proses penelusuran ini akan terus berlangsung hingga semua kemungkinan buffer yang dimulai dari baris satu dan kolom satu sudah ditemukan semuanya.

4. Proses pada langkah dua dan tiga kemudian akan diulangi untuk setiap kolom pada baris satu. Keseluruhan dari proses ini kemudian juga akan diulangi untuk panjang buffer mulai dari yang paling pendek yaitu 2 hingga ke panjang maksimum yang sudah diberikan. Proses pencarian akan dimulai dari panjang buffer 2 dan kemudian akan dilakukan pencarian yang dimulai dari setiap kolom pada baris pertama. Pencarian berikutnya akan dilakukan untuk panjang bufffer 3 dan kemudian juga akan dilakukan pencarian yang dimulai dari setiap kolom pada baris pertama. Proses pencarian ini akan terus berlanjut hingga semua kemungkinan panjang buffer sudah ditemukan.

5. Untuk meningkatkan efisiensi algoritma maka panjang minimal untuk buffer bisa dicari telebih dahulu sebelum dilakukannya pencarian untuk mengurangi jumlah pencarian yang akan dilakukan. Panjang minimal dari buffer akan sama dengan panjang minimal dari sekuens yang diberikan. Suatu buffer baru akan bisa memiliki sebuah reward ketika setidaknya panjang buffer lebih besar sama dengan panjang suatu sekuens. Oleh sebab itu panjang buffer minimum akan selalu sama dengan panjang minimum dari sekumpulan sekuens yang diberikan.
6. Karena reward dari masing-masing sekuens hanya bisa digunakan tepat sekali maka akan bisa dicari nilai maksimum reward yang mungkin diperoleh pada suatu permasalahan tersebut. Dengan mencari nilai tersebut terlebih dahulu maka proses pencarian akan bisa dibatasi sehingga bisa mempercepat waktu eksekusi. Ketika suatu buffer yang ditemukan sudah mempunyai reward yang sama dengan nilai maksimal dari reward yang bisa diperoleh maka proses pencarian bisa langsung diberhentikan seluruhnya.

Algoritma bruteforce ini bukanlah algoritma yang paling baik untuk menyelesaikan permasalahan ini. Hal tersebut dikarenakan kompleksitas dari algoritma ini yang bertambah secara eksponensial seiring dengan bertambahnya ukuran matriks dan panjang maksimal buffer yang diberikan. Di beberapa kondisi ketika panjang maksimal buffer atau ukuran matriks sangat besar maka program akan mengalami error sebelum bisa menemukan solusinya dikarenakan alokasi memori yang sudah terlalu penuh akibat pemanggilan fungsi rekursif yang cukup masif. Dalam tugas ini saya juga membuat dua program yaitu yang berbasis cli dan juga yang berbasis web. Program cli terkadang akan memiliki kecepatan eksekusi yang cenderung lebih cepat dibandingkan program web. Berdasarkan apa yang saya amati hal tersebut bisa terjadi karena program berbasis web sendiri cenderung lebih berat dibandingkan program berbasis cli sehingga alokasi penggunaan cpu dan memory pada program berbasis cli akan lebih baik dibandingkan program berbasis web.

BAB 2
SOURCE CODE

Bahasa pemrograman yang digunakan untuk menyelesaikan permasalahan dalam tugas ini adalah JavaScript/TypeScript. Source code algoritma brute force yang digunakan terdapat pada file src/web/utils/utils.ts. Algoritma bruteforce tersebut terdapat dalam class Solver. Inti dari algortima bruteforce ini terdapat di 2 method/function pada class Solver yaitu method/function ***calc*** dan ***solve***. Function ***solve*** merupakan fungsi rekursif untuk mencari semua kemungkinan jalur yang memenuhi aturan permainan yang diberikan. Sedangkan function ***calc*** merupakan fungsi yang digunakan untuk memanggil fungsi rekursif tersebut. Berikut adalah source code dari kedua fungsi tersebut:

Function calc:

```
public calc(): Solver {
  const start = performance.now();

  const lengths = this.seq.map((s) => s.length);

  const minLen = Math.min(...lengths);

  this.maxSumReward = this.calcMaxRewardSum(this.reward);

  for (let j = minLen; j <= this.bufferSize; j++) {
    for (let i = 0; i < this.kolom; i++) {
      if (!this.end) this.solve(j, [], [], true, 0, i);
    }
  }
  const end = performance.now();

  this.time = end - start;

  return this;
}
```

Function solver:

```
171   private solve(
172     buffLen: number,
173     currBuffCor: any[],
174     currBuff: string[],
175     isVertical: boolean,
176     bar: number,
177     kol: number
178   ) {
179     if (buffLen === 0) {
180       const currRew = calcReward(currBuff, this.regex, this.reward);
181
182       if (currRew !== undefined) {
183         if (currRew === this.maxSumReward) {
184           this.maxReward = currRew;
185           this.answBuff = [...currBuff];
186           this.answBuffCor = currBuffCor.slice();
187           this.end = true;
188         } else {
189           if (this.maxReward !== undefined) {
190             if (currRew && currRew > this.maxReward) {
191               this.maxReward = currRew;
192               this.answBuff = [...currBuff];
193               this.answBuffCor = currBuffCor.slice();
194             }
195           } else {
196             this.maxReward = currRew;
197             this.answBuff = [...currBuff];
198             this.answBuffCor = currBuffCor.slice();
199           }
200         }
201       }
202     } else {
203       if (currBuff.length === 0) {
204         const buffCorTemp = currBuffCor.slice();
205         const buffTemp = [...currBuff];
206
207         buffCorTemp.push([0, kol]);
208         buffTemp.push(this.matrix[0][kol]);
209
210         if (!this.end)
211           this.solve(buffLen - 1, buffCorTemp, buffTemp, isVertical, 0, kol);
212       } else {
213         if (isVertical) {
214           for (let i = 0; i < this.baris; i++) {
215             if (!currBuffCor.find(([b, k]) => b === i && k === kol)) {
216               const buffCorTemp = currBuffCor.slice();
217               const buffTemp = [...currBuff];
218
219               buffCorTemp.push([i, kol]);
220               buffTemp.push(this.matrix[i][kol]);
```

```
221
222                 if (!this.end)
223                   this.solve(
224                     buffLen - 1,
225                     buffCorTemp,
226                     buffTemp,
227                     !isVertical,
228                     i,
229                     kol
230                   );
231                 } else {
232                   const buffCorTemp = currBuffCor.slice();
233                   const buffTemp = [...currBuff];
234
235                   if (!this.end)
236                     this.solve(0, buffCorTemp, buffTemp, !isVertical, bar, kol);
237                 }
238               }
239             } else {
240               for (let i = 0; i < this.kolom; i++) {
241                 if (!currBuffCor.find(([b, k]) => b === bar && k === i)) {
242                   const buffCorTemp = currBuffCor.slice();
243                   const buffTemp = [...currBuff];
244
245                   buffCorTemp.push([bar, i]);
246                   buffTemp.push(this.matrix[bar][i]);
247
248                   if (!this.end)
249                     this.solve(
250                       buffLen - 1,
251                       buffCorTemp,
252                       buffTemp,
253                       !isVertical,
254                       bar,
255                       i
256                     );
257                   } else {
258                     const buffCorTemp = currBuffCor.slice();
259                     const buffTemp = [...currBuff];
260
261                     if (!this.end)
262                       this.solve(0, buffCorTemp, buffTemp, !isVertical, bar, kol);
263                   }
264                 }
265               }
266             }
267           }
268         }
```

Pada function solve juga digunakan function calcReward untuk menghitng nilai total reward pada suatu buffer. Berikut source code untuk function calcReward:

```
function calcReward(
  buffer: string[],
  regex: RegExp[],
  reward: number[]
): number | undefined {
  const str = buffer.join(" ");
  let rew = undefined;
  for (let i = 0; i < regex.length; i++) {
    const res = str.match(regex[i]);
    if (res) {
      if (!rew) {
        rew = reward[i];
      } else {
        rew += reward[i];
      }
    }
  }

  return rew;
}
```

Dalam fungsi calc juga digunakan sebuah fungsi lain yang bernama fungsi calcRewardSum. Fungsi tersebut digunakn untuk menghitung nilai maksimum dari reward yang mungkin didapatkan dari data yang diberikan. Berikut adalah source code dari fungsi tersebut:

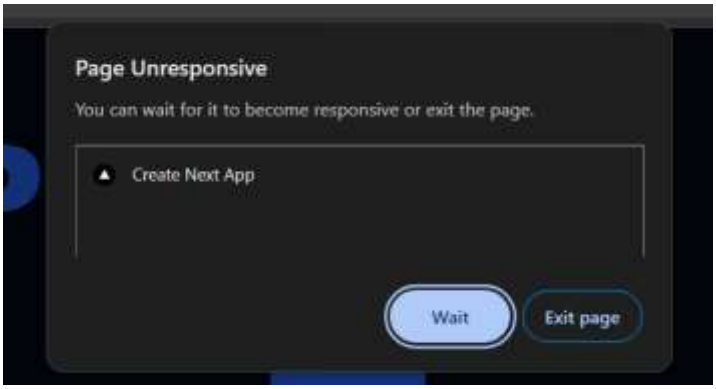
```
private calcMaxRewardSum(arr: number[]) {
  let maxSoFar = 0;

  const natNum = arr.filter((n) => n >= 0);

  if (natNum.length > 0) {
    maxSoFar = natNum.reduce((a, b) => a + b, 0);
  } else {
    maxSoFar = Math.max(...natNum);
  }

  return maxSoFar;
}
```

Program ini dibuat dalam bentuk web dan menerima 3 macam masukan. Masukan yang pertama adalah masukan biasa hampir sama seperti pada website berikut: https://cyberpunk-hacker.com/. Masukan kedua berupa file dengan format file masukan sesuai dengan spesifikasi tugas yang sudah diberikan. Masukan ketiga sama seperti masukan CLI yang diberikan pada spesifikasi tugas hanya saja diubah penampilannya ke dalam bentuk web.



Jika saat menjalankan program web mendapat pemberitahuan seperti gambar di atas ini maka sebaiknya pemberitahuan tersebut diabaikan dan tetap menunggu saja. Hal tersebut bukanlah error hanya saja terkadang untuk beberapa kasus program memerlukan waktu yang cukup lama untuk mencari solusi sehingga menimbulkan kejadian seperti gambar di atas.

Program ini juga dibuat dalam bentuk cli atau terminal sesuai dengan spek yang diberikan. Source code dari algoritma dasarnya tidak berbeda jauh dengan program yang berbentuk web mungkin hanya berbeda variabel saja. Source code pada program berbentuk cli bisa di lihat pada folder src/cli/index.js.

BAB 3

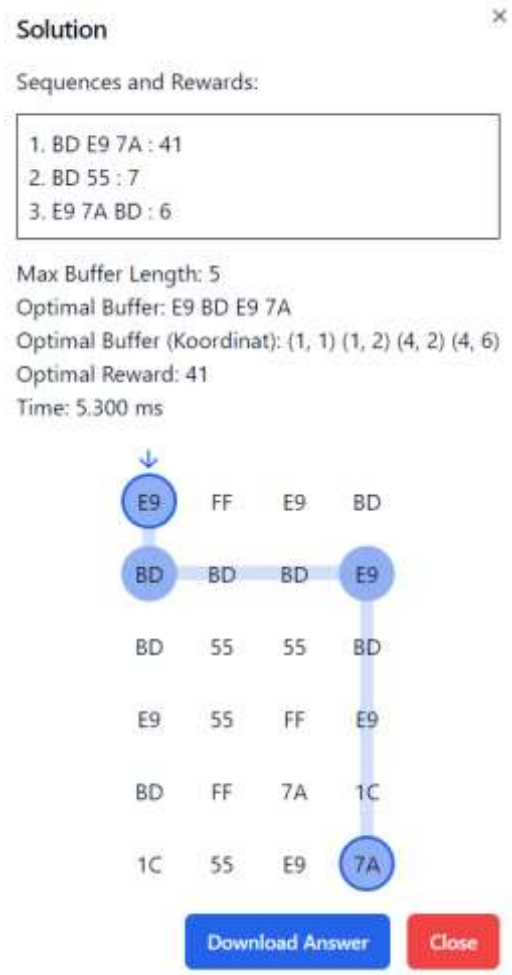
TEST CASE

1. Test Case 1

Input:

```
5
4 6
E9 FF E9 BD
BD BD BD E9
BD 55 55 BD
E9 55 FF E9
BD FF 7A 1C
1C 55 E9 7A
3
BD E9 7A
41
BD 55
7
E9 7A BD
6
```

Output:

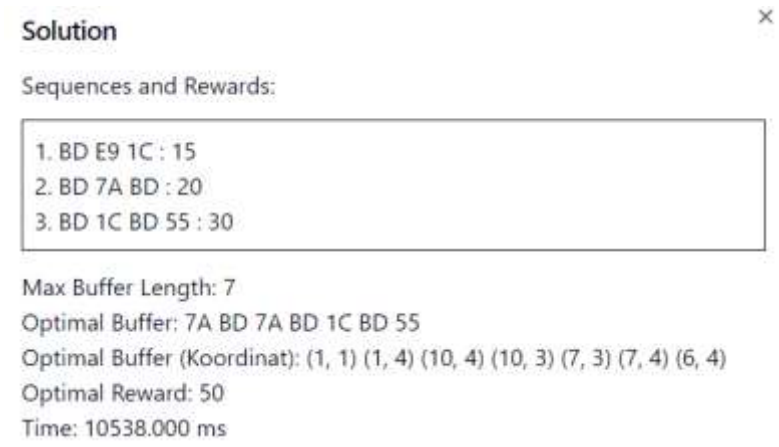


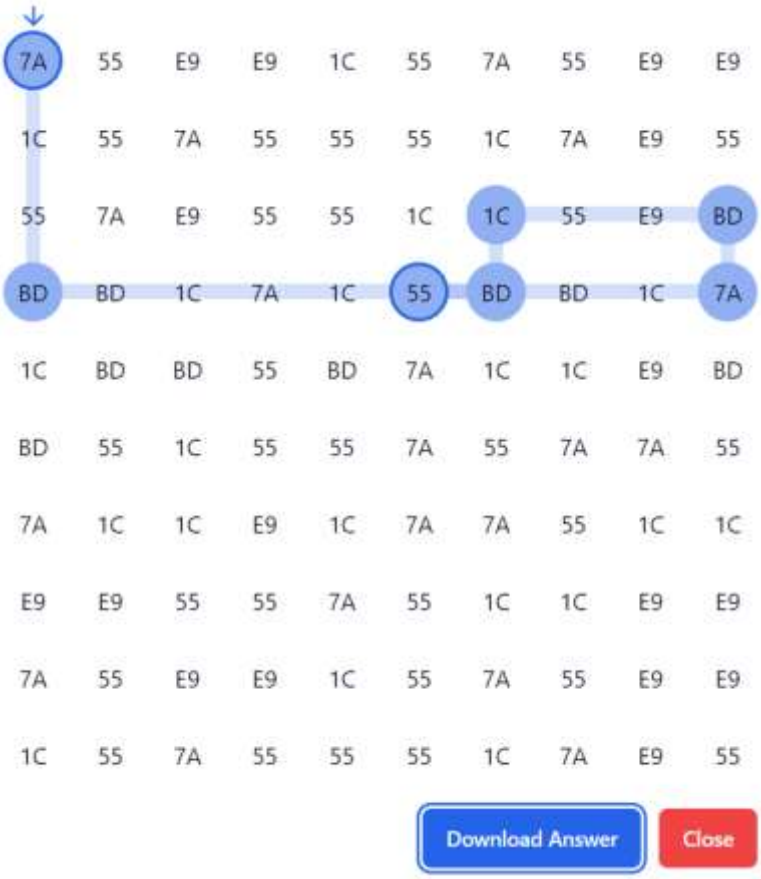
2. Test Case 2

Input:

```
7
10 10
7A 55 E9 E9 1C 55 7A 55 E9 E9
1C 55 7A 55 55 55 1C 7A E9 55
55 7A E9 55 55 1C 1C 55 E9 BD
BD BD 1C 7A 1C 55 BD BD 1C 7A
1C BD BD 55 BD 7A 1C 1C E9 BD
BD 55 1C 55 55 7A 55 7A 7A 55
7A 1C 1C E9 1C 7A 7A 55 1C 1C
E9 E9 55 55 7A 55 1C 1C E9 E9
7A 55 E9 E9 1C 55 7A 55 E9 E9
1C 55 7A 55 55 55 1C 7A E9 55
3
BD E9 1C
15
BD 7A BD
20
BD 1C BD 55
30
```

Output:





3. Test Case 3

Input:

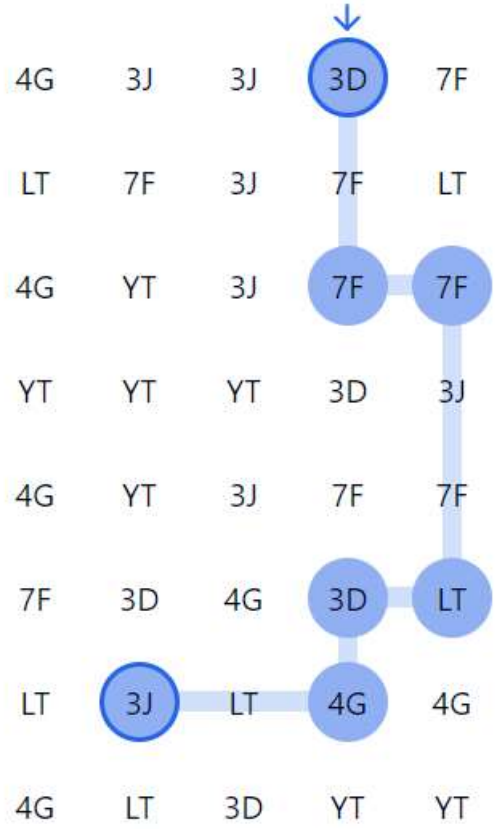
Output:

Solution

Sequences and Rewards:

1. 7F 7F LT 3D : 28
2. 4G 3J : 2
3. LT 3J 3D 3J : 52
4. LT 4G YT 3D : 21

Max Buffer Length: 8
Optimal Buffer: 3D 7F 7F LT 3D 4G 3J
Optimal Buffer (Koordinat): (4, 1) (4, 3) (5, 3) (5, 6) (4, 6) (4, 7) (2, 7)
Optimal Reward: 30
Time: 1912.000 ms



Download Answer Close

4. Test Case 4

Input:

```
8
10 8
BD 7A 7A 55 BD FF FF 1C FF 1C
7A FF 7A BD 55 FF 55 BD 1C 55
7A 7A BD E9 55 BD 55 55 1C 7A
1C 1C FF E9 E9 FF 55 FF 7A 1C
7A BD E9 BD 7A 55 1C 1C BD FF
7A E9 E9 7A 7A FF FF BD BD BD
BD 55 E9 55 1C 7A 1C 7A 7A 7A
BD FF BD FF BD FF BD 1C BD 1C
5
BD 1C 7A 1C
4
E9 55 1C 55
-29
BD FF
82
E9 7A BD FF FF
54
FF E9 E9 1C
-80
```

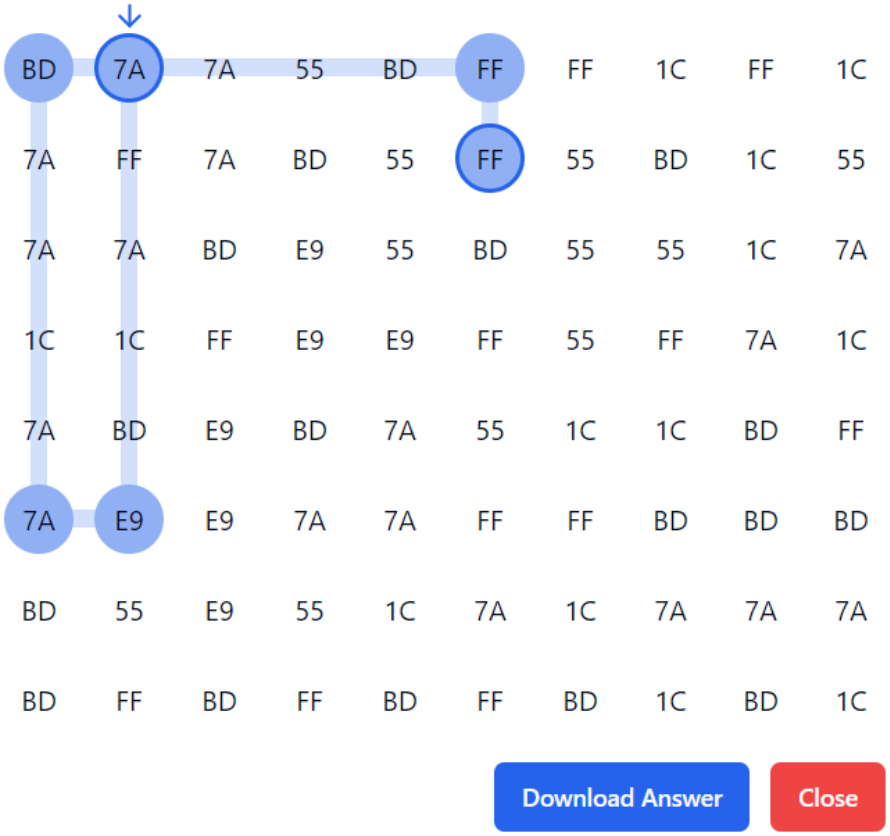
Output:

Solution

Sequences and Rewards:

1. BD 1C 7A 1C : 4
2. E9 55 1C 55 : -29
3. BD FF : 82
4. E9 7A BD FF FF : 54
5. FF E9 E9 1C : -80

Max Buffer Length: 8
Optimal Buffer: 7A E9 7A BD FF FF
Optimal Buffer (Koordinat): (2, 1) (2, 6) (1, 6) (1, 1) (6, 1) (6, 2)
Optimal Reward: 136
Time: 48977.500 ms

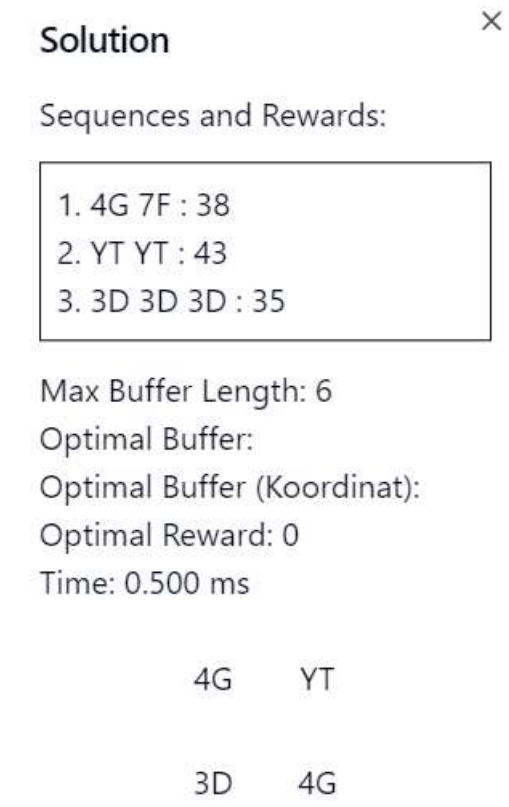


5. Test Case 5

Input:

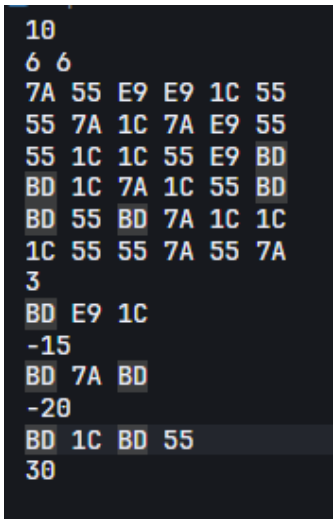
```
6
2 2
4G YT
3D 4G
3
4G 7F
38
YT YT
43
3D 3D 3D
35
```

Output:

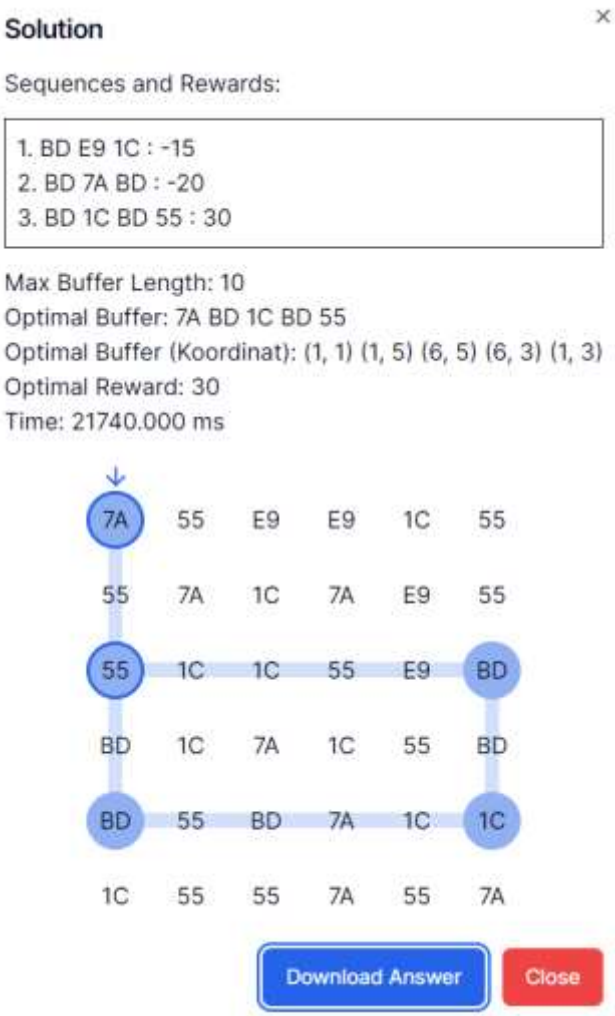


6. Test Case 6

Input:



Output:

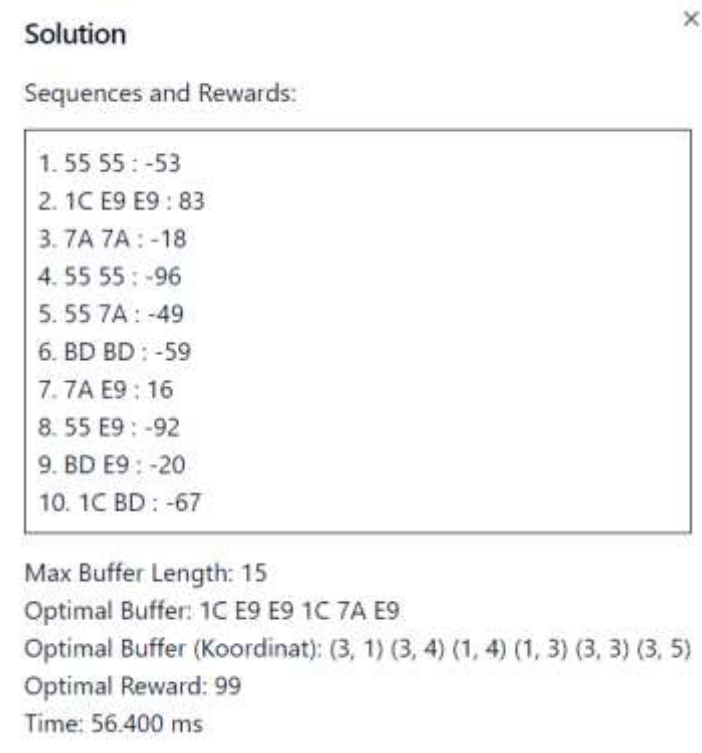

Solution
Sequences and Rewards:
1. BD E9 1C : -15
2. BD 7A BD : -20
3. BD 1C BD 55 : 30
Max Buffer Length: 10
Optimal Buffer: 7A BD 1C BD 55
Optimal Buffer (Koordinat): (1, 1) (1, 5) (6, 5) (6, 3) (1, 3)
Optimal Reward: 30
Time: 21740.000 ms
7A 55 E9 E9 1C 55
55 7A 1C 7A E9 55
55 1C 1C 55 E9 BD
BD 1C 7A 1C 55 BD
BD 55 BD 7A 1C 1C
1C 55 55 7A 55 7A
Download Answer
Close

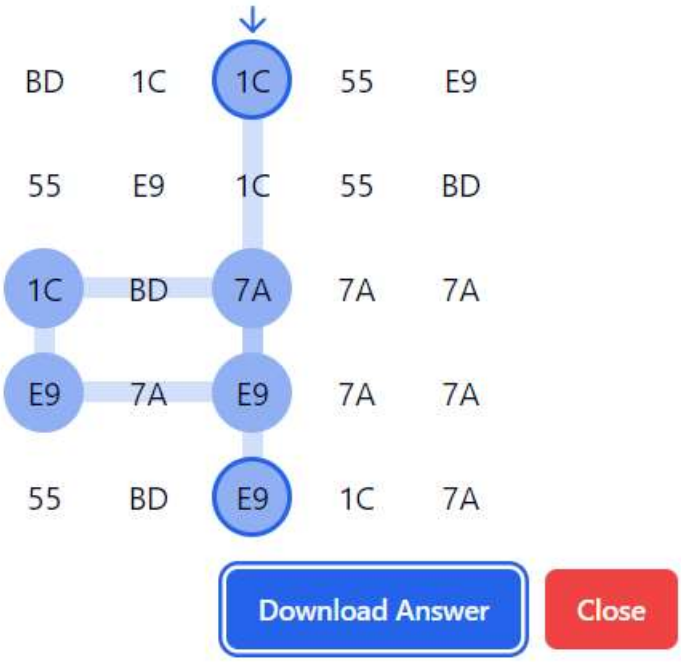
7. Test Case 7

Input:

```
15
5 5
BD 1C 1C 55 E9
55 E9 1C 55 BD
1C BD 7A 7A 7A
E9 7A E9 7A 7A
55 BD E9 1C 7A
10
55 55
-53
1C E9 E9
83
7A 7A
-18
55 55
-96
55 7A
-49
BD BD
-59
7A E9
16      You, 23 hours
55 E9
-92
BD E9
-20
1C BD
-67
```

Output:




BD 1C 1C 55 E9
55 E9 1C 55 BD
1C BD 7A 7A 7A
E9 7A E9 7A 7A
55 BD E9 1C 7A
Download Answer
Close

LAMPIRAN

Link Github: https://github.com/Otzzu/tucil-stima-satu

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	V	
2. Program berhasil dijalankan	V	
3. Program dapat membaca masukan berkas .txt	V	
4. Program dapat menghasilkan masukan secara acak	V	
5. Solusi yang diberikan program optimal	V	
6. Program dapat menyimpan solusi dalam berkas .txt	V	
7. Program memiliki GUI	V	