АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
ГМО МОРКОВИ

Решението на задачата е базирано на техниката „Динамично оптимиране“.

Първото наблюдение, което можем да направим е, че заекът има смисъл да прави смяна на посоката си на движение, само ако стигне до морков. Всяка друга смяна на посоката би го забавила излишно, защото в условието е казано, че „*целта на заека е да излезе възможно най-бързо*“.

Второто ни наблюдение е базирано на условието на задачата, в което се казва „*когато заекът попадне на позиция, на която има морков, той винаги го изяжда*“. Поради тази причина можем да използваме фактът, че тръгвайки от началната позиция морков не може да бъде достигнат, ако по пътя му не сме изядени всички други моркови. Така, движейки се от позиция A до позиция B, в какъвто и ред да сме посетили морковите, със сигурност всеки един от тях е изяден.

За решение на задачата сме приложили рекурсивен подход с меморизация (запазване на вече намерените решения, с цел да не се смятат отново). В началото след като прочетем входа, в един вектор (масив) слагаме всички обекти в Харлемшейкландия (началната позиция, крайната позиция и всички моркови). След това сортираме масива и така получаваме всички важни точки, които използваме нататък в решението.

Решението на задачата, което е дадено по-долу, използва рекурсивен метод solve, който приема като параметри скоростта на заека, два индекса в масива от обекти и връща като резултат най-краткото време, за което можем да стигнем до крайната позиция. На всяка стъпка метода проверява дали не сме стигнали вече до крайната позиция и дали не сме намерили решение вече на задачата (масива memo) и ако не сме извиква себе си за съседните моркови (наляво и надясно от текущия), към които заекът може да тръгне и увеличава скоростта двойно (заради изядения морков). Намереното минимално време от solve се запаметява в масива memo. Първото извикване на метода solve е от индекса на началната позиция в масива и скорост 1 (началната скорост на заека).

Заради използването на масива memo, никога не пресмятаме една комбинация от индекси два пъти и за това сложността на решението е квадратна (N * N).

Автор: Николай Костов

```
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <vector>
#include <algorithm>
#define MAXN 100
using namespace std;

int n;
int startPosition, endPosition;
vector<int> objects;
double memo[MAXN][MAXN];

double solve(int fromIndex, int toIndex, double speed)
{
	if (objects[fromIndex]==endPosition) return 0;
	if (objects[toIndex]==endPosition)
		return abs(objects[fromIndex]-objects[toIndex])/speed;
	if (memo[fromIndex][toIndex]!=-1) return memo[fromIndex][toIndex];
```

```
	double solution = 1000000000;
	if (0 < fromIndex && fromIndex <= toIndex)
	{
		solution = min(solution,
				(objects[fromIndex] - objects[fromIndex-1]) / speed
				+ solve(fromIndex-1, toIndex, speed*2));
	}

	if (fromIndex <= toIndex && toIndex < n - 1)
	{
		solution = min(solution,
				(objects[toIndex+1] - objects[fromIndex]) / speed
				+ solve(toIndex+1, fromIndex, speed*2));
	}

	if (0 < toIndex && toIndex < fromIndex)
	{
		solution = min(solution,
				(objects[fromIndex] - objects[toIndex-1]) / speed
				+ solve(toIndex-1, fromIndex, speed*2));
	}

	if (toIndex < fromIndex && fromIndex < n - 1)
	{
		solution = min(solution,
				(objects[fromIndex+1] - objects[fromIndex]) / speed
				+ solve(fromIndex+1, toIndex, speed*2));
	}

	memo[fromIndex][toIndex] = solution;
	return solution;
}

void input()
{
	int carrots;
	cin >> startPosition >> endPosition;
	cin >> carrots;
	for(int i = 0; i < carrots; i++)
	{
		int carrotPosition;
		cin >> carrotPosition;
		objects.push_back(carrotPosition);
	}
	n = carrots;
}

void initializeData()
{
	objects.push_back(startPosition);
	objects.push_back(endPosition);
	n += 2;
	sort(objects.begin(), objects.end());
	for(int i = 0; i < n; i++)
	{
		for(int j = 0; j < n; j++)
		{
			memo[i][j] = -1;
		}
	}
}
```

```
int main()
{
	input();
	initializeData();

	// solve
	int startIndex = 0;
	for(int i = 0; i < n; i++)
	{
		if (objects[i] == startPosition)
		{
			startIndex = i;
			break;
		}
	}
	double answer = solve(startIndex, startIndex, 1);

	// output
	printf("%.9f\n", answer);
}
```