Назив проблема: Судари

|  |  |
| --- | --- |
| Аутор: Андреј Ивашковић | Анализа: Андреј Ивашковић |

|  |  |
| --- | --- |
| Тагови: | препроцесирање, „ад-хок“ |

## Решење и анализа:

Овом задатку је могуће приступити на неколико начина различите ефикасности и “изводљивости” (под овим се мисли на тежину исправне имплементације конципираног алгоритма). У даљем тексту овог решења ће бити наведене разне технике којим се пише сажето решење (Комисија је саставила више решења, при чему нека имају око 60 редова C++ кода, а нека око 160 – поједини такмичари су имали решења чија је дужина прешла ХХХХХХ!). Један од циљева овог задатка је био провера колико су такмичари способни да пишу уредан и модуларан код.

### 30 поена

У случају је довољно да се примети да ће између свака два судара бити највише корака: у најгорем случају се Дрејко кретао од једног до другог краја матрице. Стога је симулација овог кретања прихватљиво решење чија је временска сложеност .

Иако иза овог решења не постоји никаква права “идеја”, оно се показује изузетно тешким за исправну имплементацију. Конкретно, у једном приступу бисмо морали да разматрамо све могуће “режиме” кретања и, у зависности од тога да ли се Дрејко креће ка северу, југу, западу или истоку, имали бисмо разне случајеве. Појединачно они нису нарочито тешки, али проблем настаје уколико желимо да уведемо измене у логику, након чега морају аналогне (али не идентичне!) измене да се уведу у свим случајевима – тада настају врло непријатне грешке. Зато ћемо представити опис неких уобичајених техника које сва четири случаја спајају у један општи.

Нумеришимо режиме кретања бројевима 0, 1, 2, 3 (0 – јужно, 1 – источно, 2 – северно, 3 – западно). Тада је очигледно да се након сваког судара нумерација нашег режима увећа за 1 (по модулу 4: прелаз 30).

Режим кретања може да се опише тиме што у сваком режиму постоји заједничко понашање: направићемо корак уколико је поље слободно; у супротном, мења се режим кретања јер је дошло до судара. Очигледно варира у зависности од режима кретања, те су нам неопходна два низа и , а њихови елементи су описани наредном табелом:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** |
|  | 0 | 1 | 0 | -1 |
|  | 1 | 0 | -1 | 0 |

Уз све ово уведено, један „корак“ може да се опише наредним псеудокодом:

korak(x, y):  
 | ako (zauzet[x + dx[rezim], y + dy[rezim]])  
 | | onda:  
 | | | rezim := (rezim + 1) mod 4  
 | | | broj\_sudara := broj\_sudara + 1  
 | | u suprotnom: mrdni se na (x + dx[rezim], y + dy[rezim])

### 70 поена

Очигледно је да у овом случају тражимо решење које је линеарно по променљивој . Дакле, уколико бисмо итерирали “по сударима”, између свака два судара желимо да имамо константан број операција које нам говоре до ког поља стижемо. Ово наговештава неко препроцесирање.

Приметимо да је једно Дрејково **стање** описано положајем и актуелним режимом кретања и да јединствено одређује стање након следећег судара са зидом. Конкретно, имамо неколико случајева:

* очигледно је немогуће да се Дрејко нађе „у зиду“, па нас такви случајеви не занимају;
* ако је Дрејко одмах поред неког зида и уколико је усмерен ка њему, стање ће одговарати само промењеном режиму кретања;
* у супротном, судар (тј. стање након наредног судара) до ког нас води тренутно кретање одговара судару до ког нас води поље на ком ћемо се налазити у следећем кораку (видети слику).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| ☺ |  | ☺ |  | ☺**#** |
|  |  |  |  |  |

Последња чињеница је најважнија и поприлично усложњава решење.[[1]](#footnote-1) Дакле, неопходно је одредити за сваки пар наредно стање. Ово очигледно може да се представи помоћу три низа са по три координате (координате су режим, и ), али како их израчунати? Рекурентна веза је постављена, одакле почети?

Приметимо да нам при израчунавању параметара наредног стања при фиксираном режиму уопште не утичу резултати из других режима. Дакле, одговоре за “северно”, “јужно”, “источно” и “западно” можемо да рачунамо потпуно независно. Из дефиниције режима кретања долазимо до закључака:

* наредно стање уколико је режим кретања “јужно” ће прво бити израчунато у смеру југ–север;
* наредно стање уколико је режим кретања “северно” ће прво бити израчунато у смеру север–југ;
* наредно стање уколико је режим кретања “источно” ће прво бити израчунато у смеру исток–запад;
* наредно стање уколико је режим кретања “јужно” ће прво бити израчунато у смеру запад–исток.

Дакле, можемо да попуњавамо стања у четири матрице (у неким имплементацијама можда и осам матрица – Дрејкова усмереност и врста координате на коју се односи)

Сада је уопштавање свих ових случајева нешто компликованије и једна процедура која би генерисала наредна стања при фиксном режиму би била имала нешто неразумљивији облик. Зато се предлаже да се у овом решењу “жртвује” модуларност кода зарад једноставности.

Дакле, за решење у ком се најпре препроцесирају “ланци” стања у , након чега се у размотре сви судари, предвиђено је 70 поена.

### 100 поена

Уведимо појам стања као у решењу за 70 поена. Приметимо да их нема више од : за сваки положај постоје четири режима кретања. Уколико је дат било који довољно дугачак низ стања (дужине бар ), у њему ће се неко стање сигурно наћи бар два пута (Дирихлеов принцип). Међутим, пошто је наредно стање увек унапред одређено, закључујемо да је **низ стања периодичан**.[[2]](#footnote-2)

Ова чињеница није толико необична, али нам помаже да дођемо до решења чија временска сложеност неће зависити од . Довољно је да се симулира кретање све док не стигнемо до -тог судара или не наиђемо на поље које смо већ обишли (водити рачуна о оба случаја!).

У поменутом другом случају, за симулацију може да се користи решење за 70 поена, али није нужно због природе проблема – довољно је користити просту симулацију из решења вредног 30 поена (поменутих стања нису само она када се мења режим кретања!).

Након одређивања периода и редног броја судара који је први у циклусу, уз мало модуларне аритметике се једноставно одреди завршно стање. Конкретно, уколико је редни број првог судара у циклусу , дужина циклуса , а тражимо -ти положај, тада је редни број судара који нас занима у овом циклусу . Покушајмо да оправдамо: првих корака у симулацији су пре циклуса, стога смо у току цикличног обиласка имали корака; нулто и -то стање у циклусу су идентични, те је довољно посматрати модул .

Дакле, решење за 30 поена је довољно проширити увођењем помоћних матрица које нам говоре када смо први пут дошли до неког стања. Водећи се идејом компактности записа, предлаже се да се и даље користи једна једина матрица, али ће имати три димензије – додатна координата је индекс смера! Пре него што се први пут доспе у неко стање, време неопходно да се до њега дође се најпре подеси на неко немогуће (на пример, ). При обиласку неког стања се најпре испита да ли је реч о неком раније обиђеном стању (тако што се види да ли је у том пољу поменута „немогућа“ вредност), након чега није тешко одредити период.

Опет, неопходно је обратити пажњу на случај када до траженог судара дође пре уласка у циклус!

### Статистика

Такмичарима се овај задатак показао нарочито тешким – ниједан задатак на Окружном такмичењу није слабије урађен у смислу просечног броја поена (6,36 од 100) и броја исправних решења. Ипак, тројица такмичара[[3]](#footnote-3) су остварила максималних 100 поена на овом задатку: Никола Павловић, Алекса Милојевић и Иван Дамњановић. Честитке!

Највећи не-нулти освојени број поена био је, очекивано, 30. Следи хистограм на ком се види расподела поена.

### Тестирање

Решења такмичара су тестирана на скупу од 20 тест примера, при чему је сваки био вредан по 5 поена.

### Уобичајене грешке

Такмичари су поклекли у имплементацији – био је (очекиван) огроман број компликованих решења при покушајима имплементације за 30 поена (под “компликованих” се мисли на то да није употребљена техника са ). Ипак, чак и такмичари који су успели да искористе неке делове поменутих “пречица” су направили неке друге “почетничке грешке”, посебно у виду граница низа. Конкретно, из такмичарског кода (који је такмичару донео 15 поена):

case 2:

{

while(l[y-1][x-1]==0 && x<=m)

{x++;}

s++;

x--;

break;

}

case 3:

{

while(l[y-1][x-1]==0 && y>=0)

{y--;}

s++;

y++;

break;

}

Поједини такмичари су имали исправну идеју за свих 100 поена, али су занемарили кључан специјалан случај за који се испоставило у тест примерима да нису толико “специјални” – преко половине тест примера упада у ову класу!

1. Ово је идеја која је позната при решавању задатака **динамичким програмирањем** – због тога што ова област није део градива предвиђеног за окружно такмичење, решење Комисије за 100 поена не захтева коришћење ове технике. [↑](#footnote-ref-1)
2. Постоји и математички став који се доказује коришћењем сличног резона: постоји Фибоначијев број који је дељив са (какво год било у питању). Предлажемо такмичарима да пробају сами да дођу до овог доказа имајући у виду ову „помоћ“: познавање дискретне математике и техника доказивања је кључно у рачунарству. [↑](#footnote-ref-2)
3. За три више него што се аутор задатка надао. [↑](#footnote-ref-3)