

### Задатак C13. Тунели

 1.25 s  1024 MB

Зеџ је ископао систем тунела који има  $N$  нивоа (означених од врха ка дну: од 0 до  $N - 1$ ). Сваки ниво има  $M$  одаја - соба (означене са лева на десно: од 0 до  $M - 1$ ). Одаје можемо замислити као да су поља на  $N$  пута  $M$  матрици. Постоје тунели између сваког пара суседних одаја (који деле страницу). Испод свега тога, налази се велика соба са благом (заузимајући цео ниво  $N$ ). Постоје тунели између ње (собе са благом) и свих одаја које се налазе на нивоу  $N - 1$ . Међутим, неке од  $NM$  одаја су се урушиле и због тога више нису проходне (као и сви тунели тих соба).

Зеџ жели да заштити његово благо и због тога је поставио замке у скоро све вертикалне тунеле. Прецизније, сваки ниво (укључујући и ниво у ком се налази благо) има тачно један безбедан (који не садржи замку) вертикални тунел кроз који се може доћи. Такође, гарантује се да постоји безбедан пут (пут који пролази само кроз безбедне тунеле) од врха до собе са благом.

Алиса жели да дође до собе са зечевим благом, али улазити у систем тунела без икаквих информација би било веома опасно. За почетак, она је истраживала о одајама користећи сонар и схватила је које од њих су блокиране. Међутим, она још увек нема представу у којим тунелима се налазе замке и који су безбедни. Затим је она почела да проучава зеца. Она зна да је он увек у журби, због чега увек иде најкраћим безбедним (који не садржи тунеле са замкама) путем од врха до блага (и обрнуто), тј. јединственим најкраћим путем који не посећује исту одају више пута. Она га је такође видела како улази у систем тунела (прецизније ниво 0) са површине изнад одаје  $K$  ( $0 \leq K < M$ ). Ово јој омогућава да безбедно уђе у систем тунела. За крај, Алиса је повела пса трагача који зна мириц зеца. Уз помоћ пса, она може да оде до достишне одаје (која се налази у њеном тренутном нивоу) и истражи је - провери да ли је зеџ био у тој одаји или не. Након низа истрага, када она постане сигурна, она може отићи дубље у суседну одају (у следећи ниво) из тренутне одаје. Она не жели да ризикује свој живот, због чега она мора бити апсолутно сигурна да је изабрани вертикални тунел безбедан. Она жели да стигне до блага безбедно користећи што је могуће мање истраживања у најгорем случају (она је такође у журби, јер мора да стигне на чајанку).

Формално, дефинишемо број истраживања у најгорем могућем случају једне стратегије истраживања за дати тест пример ( $N$ ,  $M$ ,  $K$  и скуп непроходних одаја) на следећи начин: за сваки могући скуп безбедних тунела, изабрати пут којим ће зеџ ићи, и покренути стратегију на сваком од ових примера (независно). Број истраживања стратегије у најгорем случају је највећи број истраживања које користи на било ком од тих случајева (претпостављајући да је стратегија валидна и безбедно долази до блага у сваком од њих; у супротном, кажемо да је број истраживања у најгорем случају те стратегија бесконачност). Ми затим излистамо све могуће стратегије и нађемо минимални могући број истраживања у најгорем могућем случају. То је највећи дозвољени број истраживања у тест примеру.

Алиса је паметна девојка, али је зецов систем тунела огроман, због чега чак ни она не може одредити оптималну стратегију истраживања. Помозите јој тако што ћете написати програм који истражује систем тунела и стиже до собе са благом користећи не више од дозвољеног броја истраживања за дати тест пример, без прола-

жења кроз непроходне одаје или пролажења кроз тунеле у којима се налазе замке.  
(Твој програм треба да задовољи све ове захтеве да би се сматрао тачним за дати  
тест пример.)

### Детаљи имплементације

Ти треба да имплементираш функцију `solve`:

```
void solve(int n, int m, int k, const std::vector<std::vector<bool>>& blocked)
```

Она ће бити позвана једном по тест примеру са  $N$ ,  $M$  и  $K$ , као и описом одаја (означено као `blocked[level][position]`). Она мора да изврши истраживање тунела (почевши од нивоа 0, одаје  $K$ ) да би стигла до нивоа на ком се налази соба са благом (ниво  $N$ ) користећи не више од најмањег могућег броја истраживања у најгорем примеру за дати тест пример. Из ове функције (и осталих функција које користиш), можеш позвати функције `investigate` и `goDeeper`:

```
bool investigate(int s)
```

Позивање функције `investigate` представља одлазак у одају  $S$  у тренутном нивоу (она мора бити достижна, тј. не сме постојати непроходна одаја на путу) и проверава да ли је зец прошао кроз ову одају на његовом путу до блага. Ова функција враћа `true`, ако је прошао, и `false`, иначе.

```
bool goDeeper(int s)
```

Позивање функције `goDeeper` представља одлазак у одају  $S$  у тренутном нивоу (она мора бити достижна) и затим се спустити у следећи ниво користећи вертикални тунел на доле. Тунел мора бити безбедан (тј. да не садржи замку) и одаја испод не сме бити непроходна. Након тога, ти настављаш твоје истраживање од одаје  $S$  на следећем нивоу (осим ако ниси стигао до нивоа  $N$ , у ком случају твоја `solve` функција прекида извршавање).

Твој код ће бити компајлиран заједно са грејдером, тако да не треба имплементирати `main` функцију, нити читати из `stdin` или писати у `stdout`. Ти такође треба да укључиш (`include`) хедер `tunnels.h`.

Грејдер некада може бити прилагодљив (али свакако детерминистички), тј. он може одлучити шта ће `investigate` функција вратити или да ли је `goDeeper` успешна, базирано на одлукама које је твој програм направио раније, без унапред фиксирања скупа безбедних тунела. Међутим, гарантује се да ће све што грејдер одлучи бити валидно поштујући неки могући скуп безбедних тунела. Грејдер може прекинути своје извршавање, уколико ти позовеш `goDeeper` и покушаш да прођеш кроз тунел са замком или ако грејдер одлучи да више није могуће да решиш дати тест пример користећи дозвољен број истраживања. Додатно, време извршавања грејдера се не рачуна у укупно време извршавање програма.

### Локално тестирање

Да би тестирао свој програм локално, локални грејдер и хедер фајл су дати. Локални грејдер није прилагодљив и не врши никаква рачунања или верификације. Он чита  $N$ ,  $M$ ,  $K$  и матрицу нула и јединица (без размака), зове твоју `solve` функцију и онда исписује сваки пут када твој програм зове `investigate` или `goDeeper` (очекујући улаз за сваки `investigate` – 0 или 1 које ће функција вратити). Теби је дозвољена модификација грејдера.

### Ограничења

- $1 \leq N, M \leq 5000$

### Подзадаци

Подзадатак	Поени	Ограничење
1	5	$M = 2$
2	21	Нема непроходних одаја.
3	18	$N, M \leq 100$
4	18	$N, M \leq 400$
5	19	$N, M \leq 1000$
6	19	Без додатних ограничења.

Ти добијаш поене за дати подзадатак, само ако успешно извршиш све тестове у њему и све остале подзадатке који су укључени у њему.

### Пример из текста задатка (Sample test)

Улаз	Интеракција
2 3 1 001 100	solve(2, 3, 1, {{0, 0, 1}, {1, 0, 0}}) goDeeper(1) investigate(2): return true goDeeper(2)

Најмањи број истраживања у најгорем случају за дати тест пример је 1, тако да би решење успешно прошло овај пример.