ЗАВДАННЯ XXII МІЖНАРОДНОЇ ОЛІМПІАДИ З ІНФОРМАТИКИ ТА РЕКОМЕНДАЦІЇ ЩОДО ЇХ РОЗВ'ЯЗАННЯ

Бондаренко В.В.

(Продовження, початок у №7, №8 за 2010 рік)

ЗАВДАННЯ ДРУГОГО ТУРУ

1. ПАМ'ЯТЬ

У грі під назвою «Пам'ять» використовується 50 карток. Кожну з літер від A до Y (ASCII-коди яких відповідно дорівнюють від 65 до 89) надруковано на лицьовому боці рівно двох карток. Перед грою всі карти змішуються випадковим чином та викладаються на стіл лицьовим боком донизу.

Під час гри Джек кожного разу перевертає дві картки лицьовим боком догори та бачить дві літери. Джек отримує від мами цукерку за кожну з 25 літер тоді, коли він перший раз бачить цю літеру одночасно на обох перевернутих картках. Наприклад, коли Джек перевернув дві картки і перший раз побачив на обох картках літеру «M», він отримує цукерку. Після цього, незалежно від того, співпадають літери на перевернутих картках чи ні, картки знову перевертаються лицьовим боком донизу. Процес гри повторюється до тих пір, поки Джек не отримає всі 25 цукерок — по одній за кожну літеру.

Вам необхідно написати процедуру play, яка грає у цю гру. Ваша процедура має викликати процедуру **faceup**(C), яку реалізовано у системі оцінювання, де *C* — ціле число від 1 до 50, що визначає номер конкретної картки, яку необхідно перевернути лицьовою стороною догори. На момент виклику процедури **faceup**(*C*) картка з номером *C* не повинна бути перевернутою лицьовим боком догори. Процедура **faceup**(*C*) повертає символ, який визначає літеру, що зображена на картці з номером *C*.

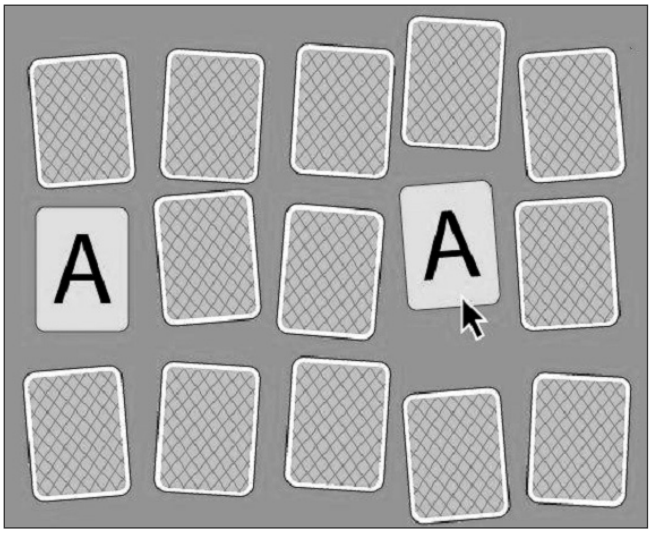

Рис. 1

Після кожного другого виклику процедури faceup система оцінювання автоматично перевертає картки знову лицьовим боком донизу.

Ваша процедура **play** може завершити свою роботу тільки тоді, коли Джек отримає всі 25 цукерок. Дозволено робити виклики процедури **faceup**(*C*) навіть після того, як Джек отримає останню цукерку.

Приклад

Нижче наведено приклад (разом з коментарями) однієї з можливих послідовностей викликів, що зроблено вашою процедурою **play**.

Таблиця 1

Виклик	Результат виклику	Коментарі
faceup(1)	'B'	На картці з номером 1 написано літеру B.
faceup(7)	'X'	На картці з номером 7 написано літеру X. Літери не співпадають.
Система оцінювання автоматично перевертає картки 1 і 7 лицьовим боком донизу.		
faceup(7)	'X'	На картці з номером 7 написано літеру X.
faceup(15)	'O'	На картці з номером 15 написано літеру O. Літери не співпадають.
Система оцінювання автоматично перевертає картки 7 і 15 лицьовим боком донизу.		
faceup(50)	'X'	На картці з номером 50 написано літеру X.
faceup(7)	'X'	На картці з номером 7 написано літеру X. Джек отримує першу цукерку.
Система оцінювання автоматично перевертає картки 50 і 7 лицьовим боком донизу.		
faceup(7)	'X'	На картці з номером 7 написано літеру X.
faceup(50)	'X'	На картці з номером 50 написано літеру X. Літери співпадають, але Джек не отримує цукерку.
Система оцінювання автоматично перевертає картки 7 і 50 лицьовим боком донизу.		
faceup(2)	'B'	На картці з номером 2 написано літеру B.
…		
(*Деякі виклики процедури були пропущені*)		
…		
faceup(1)	'B'	На картці з номером 1 написано літеру B.
faceup(2)	'B'	На картці з номером 2 написано літеру B. Джек отримує 25-ту цукерку.

Підзадача 1 [50 балів]

Реалізуйте довільну стратегію, що задовольняє правила гри та завершується у відведений час.

Наприклад, існує достатньо проста стратегія, що завжди виконує рівно 2×(49+48+…+2+1)=2450 викликів процедури **faceup(***C***)**.

Підзадача 2 [50 балів]

Реалізуйте стратегію, яка розв'язує задачу не більше ніж за 100 викликів процедури **faceup(***C***)**.

Рекомендації щодо розв'язання

Перебір всіх можливих пар карток у деякій послідовності гарантує отримання всіх 25 цукерок. Всього 50 по 2=50×49/2=1225 таких пар. Отже, вистачить 2450 перевертань карток (тобто, викликів **faceup**). Це можна зробити двома вкладеними for-циклами, та цього достатньо для розв'язання **Підзадачі 1**.

Але це не розв'язує **Підзадачу 2**, де дозволено не більше ніж 100 перевертань карток. Зауважимо що розв'язок перевертати всі пари не звертає увагу на те, що написано на перевернутих картках. Тобто, воно не використовує значень, що повертаються процедурою **faceup**. Використовуючи повернені значення, можна зібрати інформацію, що в подальшому може бути використана для зменшення кількості перевертань карток.

У крайньому випадку, можна спочатку перевернути всі картки попарно, щоб дізнатись і записати, де знаходяться всі літери, не намагаючись перевертати пари з однаковими літерами. На цьому етапі ви можете випадково отримати якісь цукерки, але це не важливо. На наступному етапі ви знаєте, де знаходяться однакові пари, та ви можете перевернути їх, послідовно, щоб отримати решту цукерок.

Перший етап потребує 50 перевертань (викликів до **faceup**), та другий етап ще 50 перевертань. Отже, разом картки перевертаються 100 разів, отже Підзадачу 2 розв'язано.

На другому етапі ви можете пропустити однакові пари, що вже було ідентифіковано на першому етапі. Однак це не покращує продуктивність у найгіршому випадку, проте ускладнює кодування.

Нижче наведено розв'язок на Pascal:

```
type
 TCard = 'A' .. 'Y'; { літери, що з'являються
                       на картках }
const
 NLetters = Ord(High(TCard)) –
   Ord(Low(TCard)) + 1; { кількість літер }
 NCardsPerLetter = 2; { кількість карток
                        на літеру }
 NCards = NCardsPerLetter * NLetters;
   { кількість карток }
 Unknown = 0; { коли індекс картки невідомо }

type
 TIndex = 1 .. NCards; { індекс картки }
 procedure play;
 var
  index: array [ TCard, 1 .. NCardsPerLetter ]
         of Unknown .. NCards;
   { index[lt, k] = індекс k-ої картки з літерою lt }
  lt: TCard; { проходить по індексу }
  k: 1 .. NCardsPerLetter; { проходить по індексу }
  i: TIndex; { проходить по картках }
  r: TCard; { результат faceup }

begin
 { ініціалізуємо індекс в Unknown }
 for lt := Low(TCard) to High(TCard) do begin
  for k := 1 to NCardsPerLetter do begin
   index[lt, k] := Unknown
  end { for k }
 end { for lt }
;
 { перший етап: не думаємо про цукерку;
   визначаємо, де знаходяться всі літери }
 for i := 1 to NCards do begin
  r := faceup(i)
 ; k := 1
 ; while index[r, k] <> Unknown do k := k + 1
 ; index[r, k] := i
 end { for i }
;
 { другий етап: тепер збираємо решту цукерок }
 for lt := Low(TCard) to High(TCard) do begin
  for k := 1 to NCardsPerLetter do begin
   r:=faceup(index[lt, k]) { результат ігноруємо }
  end { for k }
 end { for lt }
end;
```

На C, без констант та опису типів, те ж саме можна запрограмувати так:

```
void play() {
 // літери 'A' до 'Y' замінені цілими від 0 до 24
 int index[50][2]; // розташування карток;
  0 = невідомо
 int lt, k; // проходить по індексу
 int i; // проходить по індексу
 char r; // результат faceup
 // ініціалізуємо індекс
 for (lt = 0; lt < 25; ++lt) {
  for (k = 0; k < 2; ++k) {
   index[lt][k] = 0;
  }
 }
 // перший етап
 for (i = 1; i <= 50; ++i) {
  r = faceup(i);
  lt=(int)(r)–(int)('A'); // int ціле, що відповідає r
  k = (index[lt][0]) ? 1 : 0;
  index[lt][k] = i;
 }
 // другий етап
 for (lt = 0; lt < 25; ++lt) {
  faceup( index[lt][0] ); // результат ігнорується
  faceup( index[lt][1] ); // результат ігнорується
 }
}
```

2. КОРКИ НА ДОРОГАХ

Не дивлячись на велику площу Канади, багато її районів до сих пір не заселено, і більша частина населення живе поблизу південного кордону. Трансканадська магістраль, яку було побудовано у 1962 році, з'єднує людей вздовж цієї смужки землі від Сент-Джона на сході до Вікторії на заході. Загальна її довжина — 7821 км.

Рис. 2

Канадійці люблять хокей. Після хокейного матчу тисячі фанатів сідають у свої машини і їдуть додому з місця проведення матчу, що викликає великі корки на дорогах. Один заможній інвестор хоче купити хокейну команду і побудувати новий хокейний стадіон. Ваша задача — допомогти йому обрати місцезнаходження стадіону, яке б мінімізувало корки на дорогах після матчу.

Країна складається з міст, що з'єднані дорогами. Усі дороги двохсторонні, і між довільними двома містами існує рівно один *маршрут*. Маршрут, що з'єднує міста c_0 і c_k — це послідовність різних міст $c_0, \ldots, c_k$ така, що існує дорога з c_{i-1} до c_i для всіх i. Новий стадіон має бути побудований у одному з міст, назвемо його містом зі стадіоном. Після хокейного матчу всі фанати, крім тих, що живуть у місті зі стадіоном, їдуть з міста зі стадіоном у своє рідне місто. Величина корку на кожній дорозі пропорційна кількості фанатів, що проїдуть по цій дорозі. Вам необхідно обрати таке місцезнаходження стадіону, щоб величина корку на дорозі із самим великим корком була мінімально можливою. Якщо існує кілька міст, що однаково підходять для розташування стадіону, ви можете обрати довільне з них.

Вам необхідно реалізувати процедуру LocateCentre(N,P,S,D), де N — додатне ціле число, що задає кількість міст. Міста пронумеровано від 0 до (N−1). P — це масив з N додатних цілих чисел. Для кожного i елемент масиву $P[i]$ — це кількість фанатів, які мешкають у місті з номером i. Загальна кількість фанатів, що мешкають в усіх містах, буде не більше 2 000 000 000. S і D — це масиви з (N−1) цілих чисел кожен, які задають розташування доріг. Для кожного i існує дорога, що з'єднує два міста з номерами $S[i]$ і $D[i]$. Процедура має повертати ціле число — номер міста, у якому необхідно побудувати стадіон.

Приклад

Для прикладу розглянемо дорожню мережу з п'яти міст на верхньому з рисунків (рис. 3). На цьому рисунку у містах з номерами 0, 1 і 2 живуть по 10 фанатів, а у містах з номерами 3 і 4 живуть по 20 фанатів. Середній рисунок показує розмір корків, якщо стадіон буде побудовано у місті з номером 2. У цьому випадку найбільший корок розміром 40 буде досягатись на

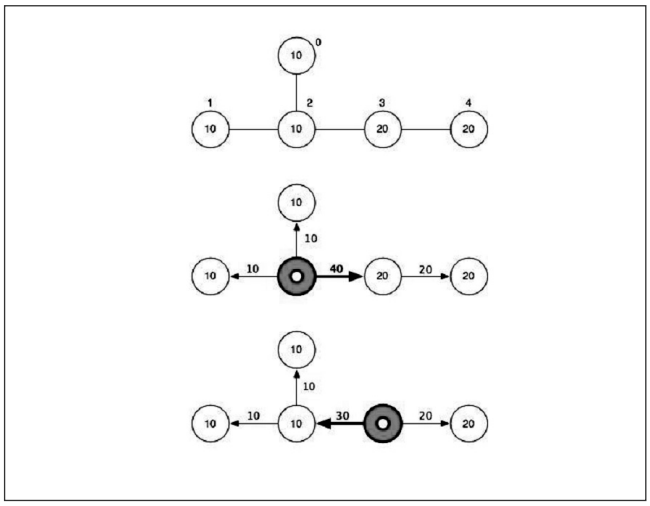

Рис. 3

дорозі, позначено жирною стрілкою. Нижній рисунок показує розмір корків, якщо стадіон буде побудовано у місті з номером 3. У цьому випадку найбільший корок розміром 30 буде досягатись на дорозі, що позначено жирною стрілкою. Таким чином, місто з номером 3 краще підходить для будівництва стадіону, ніж місто з номером 2.

Зауваження

Ми нагадуємо учасникам, що за даних обмежень можна послати розв'язок, що проходить підзадачу 3 і не проходить підзадачу 2. Однак, пам'ятайте, що ваші фінальні очки по задачі визначаються *тільки одним* з ваших посилань.

Підзадача 1 [25 балів]

Припустимо, що всі міста лежать на прямій від сходу до заходу, і всі дороги йдуть по прямій без розгалуження. Більш формально, припустимо, що для всіх i з діапазону $0 \le i \le N-2$ виконуються співвідношення $S[i]=i$ і $D[i]=i+1$.

Кількість міст не перевищує 1000.

Підзадача 2 [25 балів]

Розв'яжіть задачу при тому ж самому припущенні, що і в підзадачі 1, але кількість міст може досягати 1 000 000.

Підзадача 3 [25 балів]

Припущення з підзадачі 1 більше не виконується. Кількість міст не перебільшує 1 000.

Підзадача 4 [25 балів]

Припущення з підзадачі 1 більше не виконується. Кількість міст не перевищує 1 000 000.

Рекомендації щодо розв'язання

Ця задача є достатньо стандартною. Хоча треба зауважити, що графові задачі зазвичай виявляються трохи складнішими, ніж здаються на перший погляд, оскільки треба обробляти якесь специфічне кодування графа. Підзадачі потребують алгоритмів такої складності.

Підзадача 1. Буде працювати квадратичний алгоритм. Завдяки дуже регулярній (лінійній) структурі графа мережі, можна просто спробувати кожне з міст як місце розташування арени, обчислити найбі-

льше скупчення та вибрати розташування, де найбільше скупчення є мінімальним.

Підзадача 2. Потребує лінійного алгоритму, але оскільки є тільки 2 листи і представлення графу є дуже регулярним, просто помітити, що одного проходу по містах вздовж дороги достатньо для визначення оптимального розташування.

Підзадача 3. Буде працювати квадратичний алгоритм, але потрібно обробляти довільний граф. Знову, як в підзадачі 1, кожне місто має бути випробуваним як місце розташування арени, знайдено найгірший корок, після чого можна знайти найкраще розташування арени.

Підзадача 4. Це повна задача. Лінійний прохід по графу, що накопичує відповідним чином інформацію про корки, дозволяє визначити оптимальне розташування арени за лінійний час.

3. ЛАБІРИНТ

У південному Онтаріо багато фермерів, що вирощують кукурудзу, створюють на полі лабіринти із стрижнів кукурудзи, один з яких наведено на рисунку. Лабіринти створюються восени, коли урожай качанів уже зібрано. У вас ще є час допомогти розробити найкращий лабіринт для 2010 року.

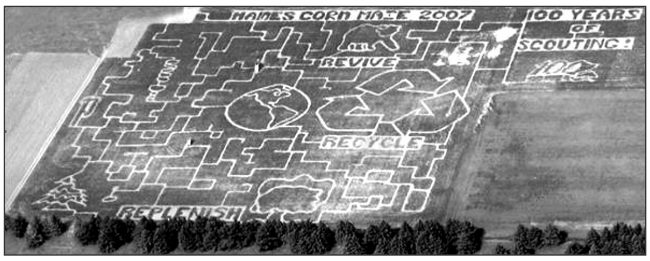

Рис. 4

Поле засіяно кукурудзою за виключенням декількох перешкод (дерев, будівель і тому подібного), де кукурудза не росте. Стрижні, висота яких є величезною, утворюють стінки лабіринту. Стежки прокладаються на прямокутній сітці у клітинах площею у 1 квадратний метр кожна шляхом вирубання у них стрижнів. Одна така клітина сітки на межі є входом до лабіринту. Іще одна клітина у лабіринті є метою лабіринту.

Джек відвідує кукурудзяні лабіринти кожен рік, тому він став експертом у швидкому знаходженні шляху від входу до мети. Ви розробляєте новий лабіринт, і ваша задача полягає у тому, щоб визначити, у яких клітинах треба вирубати стрижні, щоб максимізувати кількість клітин у лабіринті, які має пройти Джек.

Система оцінювання визначить, яка клітина є входом (єдина з клітин на межі) і яка клітина є метою — та, до якої Джек має йти якомога довше.

Карта прямокутного поля подається у вигляді тексту, наприклад, поле розміром 6×10 метрів, що містить вісім дерев, може бути подано так:

```
# # X # # # # # # #
# # # X # # # # # #
# # # # X # # X # #
# # # # # # # # # #
# # X X X X # # # #
# # # # # # # # # #
```

Символ «#» задає клітину зі стрижнями кукурудзи, а символ «X» задає клітину з перешкодою (наприклад, деревом), що не можна видалити, щоб продовжити стежку.

Поле перетворюється у лабіринт вирубанням квадратних клітин, що засіяно кукурудзою. Одна вирубана клітина (вхід) має бути розташованою на краю поля. Решта вирубаних клітин мають бути всередині поля. Ваша задача — максимізувати найкоротший шлях путь від входу до мети, який вимірюється кількістю вирубаних клітин, що має пройти Джек, включаючи клітини входу та мети. Дозволяється переходити з однієї клітини у другу тільки тоді, коли обидві ці клітини вирубані і мають спільну сторону.

У файлі, що висилається на перевірку, вирубані клітини мають бути позначені символами «.» (точки). Рівно одна вирубана клітина має бути на межі поля. Наприклад,

```
# . X # # # # # # #
# . # X # . . . # #
# . . . X # . X . #
# . # . . . . . . #
# . X X X X # # . #
# # # # # # # # # #
```

Нижче, тільки з метою зручності ілюстрування, ми позначили вхід до лабіринту символом «E», мету лабіринту — символом «C», а решту шляху — символами «+». Довжина шляху у цьому випадку дорівнює 12 клітинам.

```
# E X # # # # # # #
# + # X # C + . # #
# + + + X # + X . #
# . # + + + + . . #
# . X X X X # # . #
# # # # # # # # # #
```

Папка /home/ioi2010-contestant/maze містить декілька текстових файлів з іменами field1.txt, field2.txt і т. д., які містять описи карт кукурудзяних полів. Ви маєте скопіювати їх у файли з іменами maze1.txt, maze2.txt і т. д. і перетворити їх у допустимі лабіринти, замінивши деякі символи «#» на символи точки («.»).

Зауваження. Система оцінювання у якості публічного тесту буде давати 1 бал за кожну підзадачу при здачі любого допустимого розв'язку по підзадачі (незалежно від довжини шляху). Решта балів буде даватись як реліз-тест. Загальна кількість балів по задачі буде заокруглюватись до найближчого цілого числа між 0 і 110.

Підзадача 1 [до 11 балів]

Описане вище поле знаходиться у файлі field1.txt (поле розміром 6×10). Створіть для цього поля у файлі з іменем maze1.txt лабіринт, який має найкоротший шлях довжини P від входу до мети. Ваш розв'язок для цієї підзадачі буде оцінено мінімумом з 11 і $10^{P/20}$ балів. Розв'язок з прикладу оцінюється у 3.98 балів.

Підзадача 2 [до 11 балів]

Файл field2.txt задає поле розміром 100×100. Створіть для цього поля у файлі з іменем maze2.txt лабіринт, який має найкоротший шлях довжини P від вхо-

ду до мети. Ваш розв'язок для цієї підзадачі буде оцінено мінімумом з 11 і $10^{P/4000}$ балів.

Підзадача 3 [до 11 балів]

Файл field3.txt задає поле розміром 100×100. Створіть для цього поля у файлі з іменем maze3.txt лабіринт, який має найкоротший шлях довжини P від входу до мети. Ваш розв'язок для цієї підзадачі буде оцінено мінімумом з 11 і $10^{P/4000}$ балів.

Підзадача 4 [до 11 балів]

Файл field4.txt задає поле розміром 100×100. Створіть для цього поля у файлі з іменем maze4.txt лабіринт, який має найкоротший шлях довжини P від входу до мети. Ваш розв'язок для цієї підзадачі буде оцінено мінімумом з 11 і $10^{P/4000}$ балів.

Підзадача 5 [до 11 балів]

Файл field5.txt задає поле розміром 100×100. Створіть для цього поля у файлі з іменем maze5.txt лабіринт, який має найкоротший шлях довжини P від входу до мети. Ваш розв'язок для цієї підзадачі буде оцінено мінімумом з 11 і $10^{P/5000}$ балів.

Підзадача 6 [до 11 балів]

Файл field6.txt задає поле розміром 11×11 Створіть для цього поля у файлі з іменем maze6.txt лабіринт, який має найкоротший шлях довжини P від входу до мети. Ваш розв'язок для цієї підзадачі буде оцінено мінімумом з 11 і $10^{P/54}$ балів.

Підзадача 7 [до 11 балів]

Файл field7.txt задає поле розміром 20×20. Створіть для цього поля у файлі з іменем maze7.txt лабіринт, який має найкоротший шлях довжини P від входу до мети. Ваш розв'язок для цієї підзадачі буде оцінено мінімумом з 11 і $10^{P/33}$ балів.

Підзадача 8 [до 11 балів]

Файл field8.txt задає поле розміром 20×20. Створіть для цього поля у файлі з іменем maze8.txt лабіринт, який має найкоротший шлях довжини P від входу до мети. Ваш розв'язок для цієї підзадачі буде оцінено мінімумом з 11 і $10^{P/95}$ балів.

Підзадача 9 [до 11 балів]

Файл field9.txt задає поле розміром 11×21. Створіть для цього поля у файлі з іменем maze9.txt лабіринт, який має найкоротший шлях довжини P від входу до мети. Ваш розв'язок для цієї підзадачі буде оцінено мінімумом з 11 і $10^{P/104}$ балів.

Підзадача 10 [до 11 балів]

Файл fieldA.txt задає поле розміром 200×200. Створіть для цього поля у файлі з іменем mazeA.txt лабіринт, який має найкоротший шлях довжини P від входу до мети. Ваш розв'язок для цієї підзадачі буде оцінено мінімумом з 11 і $10^{P/7800}$ балів.

Рекомендації щодо розв'язання

Це складна задача. Для неї не відомий загальний поліноміальний алгоритм розв'язання тобто не знайдено алгоритму, що гарантовано розв'язує кожен екземпляр задачі за поліноміальний від розміру задачі (площі кукурудзяного поля) час.

Це також пояснює, чому цю задачу було запропоновано як задачу з відкритими тестами, коли учасникам надається 10 конкретних випадків для розв'язання. Розв'язувати можна вручну або написати програми, що будуть аналізувати лабіринти та написати інші програми (можливо різні для кожного поля), щоб побудувати найдовший шлях. Зауважте, що шляхи не мають бути оптимальними, достатньо гарного наближення, щоб отримати хороші бали.

Нижче наведено деякі характеристики тестів та результати одного з членів наукового комітету(табл. 2).

Таблиця 2

	Характеристика		Результат	
Приклад	**Розміри**	**Перешкод**	**Довжина шляху**	**Бали**
1	10x10	8	20	10.00
2	100x100	1766	4026	10.15
3	100x100	3216	3740	8.61
4	100x100	2283	3733	8.58
5	100x100	1357	4738	8.86
6	11x11	0	54	10.00
7	20x20	210	33	10.00
8	20x20	122	95	10.00
9	11x21	10	106	10.45
10	200x200	15224	7506	9.17
Всього				**95.82**

4. ЗБЕРЕЖЕННЯ ІНФОРМАЦІЇ

Служба доставки «Скедеф» перевозить повітрям посилки між кількома містами. Якісь з цих міст є вузлами, і в них встановлено спеціальні пристрої для обробки посилок. Кожен з літаків компанії «Скедеф» літає в обидва боки між одною з пар міст, і перевозить посилки у довільному з двох напрямків, якщо це потрібно.

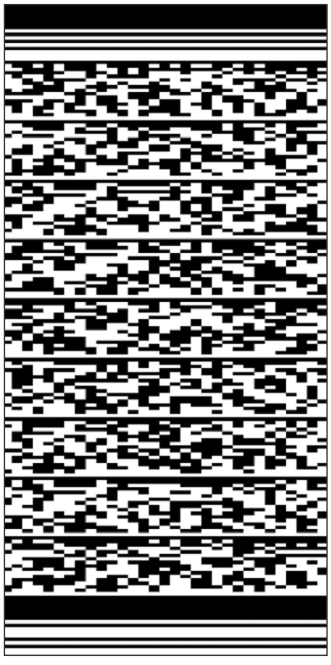

Рис. 5

Щоб перевезти посилку з одного міста в інше, її необхідно провезти, використовуючи послідовні перельоти через міста, при цьому на кожному перельоті посилка перевозиться між парою міст, що обслуговується одним з літаків. Більш того, у цій послідовності міст має бути принаймні один вузол.

Для полегшення підбору маршруту, компанія «Скедеф» хоче закодувати довжини найкоротших послідовностей перельотів і кожного міста у кожен вузол і записати коди на ярлику кожної з посилок. Довжина найкоротшої послідовності перельотів з вузла до себе дорівнює нулю. Очевидно, що потрібно компактне подання цієї інформації.

Вам необхідно реалізувати дві процедури — **encode(*N*,*H*,*P*,*A*,*B*)** і **decode(*N*,*H*)**. Тут N — кількість міст, H — кількість вузлів. Міста пронумеровані від 0 до (N−1), а вузлами є міста з номерами від 0 до

($H-1$). Крім того, $N \le 1000$ і $H \le 36$. P — кількість пар міст, між якими літають літаки. Усі пари міст є різними як невпорядковані пари. A і B — це масиви розміру P, і ($A[0]$, $B[0]$) — це перша пара міст, між якими літає літак, ($A[1]$, $B[1]$) — друга пара і т. д.

Процедура **encode** має побудувати таку послідовність бітів, за якою процедура **decode** зможе відновити кількість перельотів, необхідних для перевезення посилки від кожного з міст до кожного з вузлів. Процедура **encode** буде передавати послідовність бітів системі оцінювання за допомогою послідовних викликів процедури **encode_bit(*b*)**, де b дорівнює 0 або 1. Процедура **decode** буде отримувати послідовність бітів від системи оцінювання за допомогою викликів процедури **decode_bit**. При цьому i-й виклик процедури **decode_bit** буде повертати значення **b**, передане i-му виклику **encode_bit(b)**. Зауважимо, що ви маєте впевнитись у тому, що кількість викликів процедурою **decode** процедури **decode_bit** не буде перевищувати кількості зроблених процедурою **encode** викликів процедури **encode_bit(*b*)**.

Після декодування інформації про кількість необхідних перельотів, процедура **decode** має викликати процедуру **hops(*h*,*c*,*d*)** для кожного з вузлів h і кожного з міст c (у тому числі і для вузлів, і у випадку $c=h$), повідомляючи мінімальну кількість необхідних перельотів d, що потрібно для перевезення посилки між вузлом h і містом c. Таким чином, ви маєте зробити $N \times H$ викликів процедури **hops(*h*,*c*,*d*)**. Порядок цих викликів не важливий. Гарантується, що завжди можна перевезти посилку між довільним вузлом і довільним містом.

Увага. Процедури **encode** і **decode** мають взаємодіяти тільки через описаний вище інтерфейс. Спільне використання тих самих змінних, доступ до файлів і до мережі є забороненими.

Приклад

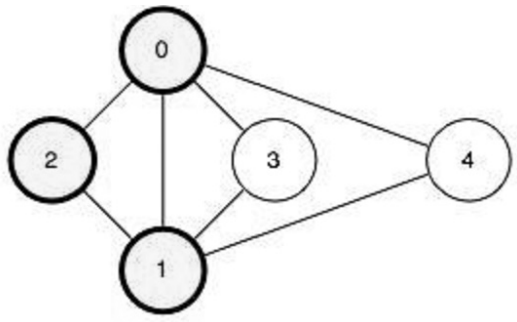

Рис. 6

Для прикладу розглянемо рисунок 6. На ньому показано п'ять міст ($N=5$), що зв'язані сімома літаками ($P=7$). Міста з номерами 0, 1 та 2 є вузлами ($H=3$). Для доставки посилки між вузлом з номером 0 і містом з номером 3 необхідно один переліт, а для доставки посилки між вузлом з номером 2 і містом з номером 3 необхідно 2 перельоти.

Усі значення d, які процедура **decode** має передати процедурі **hops(*h*,*c*,*d*)**, наведено у таблиці 3.

Підзадача 1 [25 балів]

encode повинна робити не більше 16 000 000 викликів **encode_bit(*b*)**.

Підзадача 2 [25 балів]

encode повинна робити не більше 360 000 викликів **encode_bit(*b*)**.

Таблиця 3

d		Місто c				
		0	**1**	**2**	**3**	**4**
Вузол h	**0**	0	1	1	1	1
	1	1	0	1	1	1
	2	1	1	0	2	2

Підзадача 3 [25 балів]

encode повинна робити не більше 80 000 викликів **encode_bit(*b*)**.

Підзадача 4 [25 балів]

encode повинна робити не більше 70 000 викликів **encode_bit(*b*)**.

Рекомендації щодо розв'язання

Задача такого типу є новою для IOI. Зазвичай для більшості задач IOI важливою є ефективність. Однак у цьому випадку це не час виконання або використання пам'яті, це скоріш ефективність спілкування: як подати деякі складні дані якомога меншою кількістю бітів, без втрати інформації.

Ця різниця у фокусі робить задачу можливо більш складною для розуміння. Більш того, вона є технічно більш складною, адже учасник має запрограмувати дві незалежні процедури, що інвертують одна одну. Формат взаємодії не задається умовою. Єдине, що є важливим, це те, що програма-декодер, що написав учасник, має розкодувати дані від програми-кодувальника, що написав той самий учасник.

Важливі дві речі. По-перше, знайти шлях кодування інформації про зв'язки у транспортній мережі Скедеф. По-друге, ефективно передати цю інформацію.

Стисло кажучи, підзадачі вимагають таких підходів у розв'язанні.

Підзадача 1. Можна передати всю матрицю суміжності як є. Ця інформація природно виражається у бітах, можливі й інші кодування. Усе, що мають узгодити кодувальник і декодувальник — це послідовність бітів. Оскільки є всього 1000 міст, потрібно не більше ніж $1000 \times 1000 = 1\,000\,000$ бітів. Інші підходи, що використовують більше бітів, також можуть набрати бали у цій підзадачі.

Підзадача 2. Ви можете надіслати всю таблицю з кількостями перельотів. Оскільки є не більше 1000 міст, максимальна кількість перельотів буде менше 1000, отже, може бути закодованою 10 бітами. Максимальний розмір таблиці — $1000 \times 36 = 36000$. Отже, потрібно не більше 360 000 бітів.

Підзадача 3. Потрібно використати нову ідею, щоб підвищити ефективність взаємодії. Можна застосувати остовне дерево, можна використовувати довільне остовне дерево. Відстань між $v1$ та $v2$ є одним з:

- відстань від батька ($v1$) до $v2$;
- 1 + відстань від батька ($v1$) до $v2$;
- −1 + відстань від батька ($v1$) до $v2$.

Усе, що треба зробити, це закодувати кожну з можливостей за допомогою двох бітів.

Підзадача 4. Використання двох бітів для запису трьох можливостей не є ефективним. Можливо відобразити 3 трійкових варіанти (27 варіантів) на 5 бітів (32 можливості). Це ще покращує взаємодію.