додаток 2

Готуємось до олімпіади

(додаток 2)

1.Дано послідовність з N чисел, котра містить різні числа від 0 до N. Визначити, якого числа не існує в даній послідовності.

1 спосіб.

Посортувати і відшукати різницю, рівну два між сусідніми елементами.

#include	"stdafx.h"

#include	"iostream"

using	namespace	std;

int	main()

{	int	n,i,j,temp,r,a[1000];

```
cin>>n;

for (i=1;i<=n;i++)cin>>a[i];

for (i=1;i<n;i++)

for (j=1;j<n;j++)

if (a[j]>a[j+1]){temp=a[j];a[j]=a[j+1];a[j+1]=temp;}

a[n+1]=n+1;

for (i=1;i<=n;i++)

    if (a[i+1]-a[i]==2) r=a[i]+1;

cout<<r<<endl;

return 0;

}
```

2 спосіб.

Перевірити, чи існує кожне з чисел від 0 до N у послідовності, використовуючи два вкладених ци

```
#include "stdafx.h"

#include "iostream"

using namespace std;

int main()

{ int n,i,j,s,r,a[1000];

cin>>n;

for (i=1;i<=n;i++)cin>>a[i];

for (i=0;i<=n;i++)

{s=0;

for (j=1;j<n;j++)

if (j==a[i]) s=1;
```

if	(s==0) r=i;	

}

cout<<r<<endl;

	return	0;

}

3 спосіб.

Скористатися формулою суми арифметичної прогресії.

Приклад:

N=5;

Послідовність A[1..N] 4 2 3 0 5

Сума елементів послідовності рівна S1=4+2+3+0+5=14

Сума арифметичної прогресії (0..N) 0 1 2 3 4 5 згідно з формулою

Результат R=S2-S1=15-14=1

Отже, не існує числа 1.

```
#include "stdafx.h"

#include "iostream"

using namespace std;

int main()

{ int n,i,j,s1,s2,r,a[1000];

cin>>n;

for (i=1;i<=n;i++)cin>>a[i];

s1=0;

for (i=1;i<=n;i++)s1=s1+a[i];

s2=(n+0)*(n+1)/2;
```

```
r=s2-s1;
```

```
cout<<r<<endl;
```

```
        return  0;
```

```
}
```

По вигляду, структурі і кількості простих операцій видно, що найоптимальнішими способом є останній, де використовується формула знаходження суми арифметичної прогресії.

2. Проходження лабіринту.Написати алгоритм пошуку виходу в лабіринті.

*Наявна матриця розміру M*N, що являє собою деякий лабіринт. Одна клітинка лабіринту є входом а інша виходом. Знайти найкоротший шлях від входу до виходу, якщо він існує.*

***Вхідні данні** : файл Labirint.dat, де першим рядком йде два числа N та M (0<N,M<100), а далі M рядків по N чисел відокремлених пробілами. Всі числа Amn можуть приймати значення 1,2,3,4. 1 - ділянка, що можна пройти. 2 - стіна. 3 - вхід. 4 - вихід.*

***Вихідні данні**: число T>0, яке дорівнює довженні найкоротшого шляху, або -1, якщо такого шляху не існує.*

Аналіз задачі.

По-перше треба відмітити, що максимальний розмір лабіринту, не перевищує 10^4 елементів, тобто весь набір даних можна розмістити в статичній пам'яті.

Одним з найбільш ефективних алгоритмів пошуку найкоротшого шляху є алгоритм хвилі. Хвильовий алгоритм полягає в наступному:

1. *кожної клітинок і приписується число T - тимчасова мітка.*
2. *заводяться два списки "фронту хвилі" NF і OF, а також перемінна T (поточний час);*
3. *OF:={u1}; NF:={}; T:=1;*
4. *для кожної з вершин i, що входять у OF, проглядаються сусідні вершини j, і якщо T[j] = -2, то T[j]=T, NF=NF + {j}.*
5. *якщо NF = {}, то шлях не існує, перехід до кроку 8.*
6. *якщо одна з вершин збігається з u2, то знайдений найкоротший шлях довжини T, перехід до кроку 8;*
7. *OF:=NF; NF:={}; T:=T+1; повернення до кроку 4.*
8. *Відновлюємо шлях проходячи масив P, шукая серед сусідів даної клітинки таку, щоб номер ітерації хвилі у неї був найменшим.*

1. Якось, пам'ятається, ще в грі "НЛО-1", проскочило побажання до тих, хто хоче стати добрим програмістом, підвищувати, підвищувати і ще раз підвищувати свій освітній рівень. На що із сторінок одного дуже поважаного видання виступив читач з наступною думкою (дослівно не пам'ятаю): "Я людина темна, але код програми - що треба. Значить, я вже добрий програміст". Дана стаття є спроба розвіяти цю глибоку помилку. В першу чергу вона адресована тим, хто займається створенням ігор і тим, кому не дає спокою думка: "А що там всередині?" Для її розуміння достатньо знання що таке двомірні масиви.

Побудова найкоротшого маршруту

Задача знаходження найкоротшого шляху між якимись точками A і B на ігровому полі з довільно розташованими перешкодами характерна, в першу чергу, для популярних сьогодні тактичних і стратегічних ігор. Як підзадача, вона може виникати практично в будь-яких іграх - RPG, квестах, логічних (типовий приклад "Color Lines").

Чому треба шукати найкоротший маршрут? В деяких іграх, наприклад "НЛО-2", "Laser

Squad", від довжини маршруту залежить кількість витрачених одиниць часу - чим оптимальніше буде знайдений шлях, тим швидше воїн дістанеться до мети. А, наприклад, в "Color Lines" довжина маршруту не обумовлена правилами, має значення лише сам факт можливості або неможливості переміщення кулі. Але і в цій грі приємніше виглядатиме, якщо кулька відразу попрямує куди треба, а загадково не дефілюватиме по всій ігровій дошці.

Рішення цієї задачі прийшло до нас з такої далекої, здавалося б, від ігор області як електроніка. А саме - розводка друкарської плати (всі знають, що це таке? ;).

Існує велика кількість трасувальників (програм для розводки плати), заснованих на не меншій кількості різних методів, що займаються з'єднанням двох контактів єдиним провідником. Але ми розглянемо тільки один з них, найпростіший (а значить, найнадійніший і найпопулярніший) - хвильовий трасувальник.

Поставимо перед хвильовим трасувальником задачу в термінах що розробляється нами гри:
Є ігрове поле P(MxN),де M і N, відповідно, розмір поля по вертикалі і горизонталі. Просто, це масив розмірністю MxN. Кожна клітка ігрового поля (елемент масиву) може мати багато властивостей на ваш розсуд, але для нас важливо тільки одне - її прохідність (або непрохідність). Яким чином вона визначається - це вже ваша справа. Далі: є деяка стартова точка, де знаходиться герой вашої гри, і кінцева точка, куди йому необхідно потрапити. Умовимося поки, що ходити він може тільки по чотирьох напрямах (чого вимагає від нас класичний хвильовий метод) - вправо, вліво, вперед, назад. Необхідно перемістити героя від місця старту до фінішу за якнайменшу кількість ходів, якщо таке переміщення взагалі можливо.

1. Спочатку необхідно створити робочий масив R(MxN), рівний за розміром масиву ігрового поля P(MxN).
2. Кожному елементу робочого масиву R(i,j) привласнюється деяке значення в поля P(i,j) за наступними правилами:
   а) Якщо поле P(i,j) непрохідне, то R(i,j):=255;
   б) Якщо поле P(i,j) прохідне, то R(i,j):=254;
   в) Якщо поле P(i,j) є цільовою (фінішної) позицією, то R(i,j):=0;
   г) Якщо поле P(i,j) є стартовою позицією, то R(i,j):=253.
3. Етап "Розповсюдження хвилі". Вводимо змінну Ni - лічильник ітерацій (повторів) і надаємо їй початкове значення 0.
4. Вводимо константу Nk, котору встановлюємо рівній максимально можливому числу ітерацій (

Nk
<253
).

5. По рядках проглядаємо робочий масив R (організуємо два вкладених цикла: по індексу масиву i від 1 до M,
по
i
ндексу
масиву j від 1 до N).

6. Якщо R(i,j) рівний Ni, то є видимими сусідні елементи R(i+1,j), R(i-1,j),R(i,j+1), R(i,j-1) за наступним правилом (як приклад розглянемо R(i+1,j)):
а) Якщо R(i+1,j)=253, то переходимо до пункту 10;
б) Якщо R(i+1,j)=254, виконується присвоєння R(i+1,j):=Ni+1;
в) У всій інших випадках R(i+1,j) залишається без змін. Аналогічно поступаємо з елементами R(i-1,j), R(i,j+1),R(i,j-1).
7. По завершенню рядкового перегляду всього масиву збільшуємо Ni на 1.
8. Якщо Ni>Nк,то пошук маршруту визнається невдалим. Вихід з програми.
9. Переходимо до пункту 5.
10. Етап побудови маршруту переміщення. Привласнюємо змінним X і У значення координат стартової позиції.
11. Навколо позиції R(X,Y) шукаємо елемент з якнайменшим значенням (т.т. для цього проглядаємо R(X+1,Y), R(X-1,Y), R(X,Y+1), R(X,Y-1). Координати цього елемента заносимо в змінні X1 і Y1.
12. Робимо переміщення об'єкту по ігровому полю з позиції [X,Y] в позицію [X1,Y1]. (За бажанням, ви можете заздалегідь заносити координати X1,Y1 в деякий масив, і, тільки закінчивши побудову всього маршруту, зайнятися періміщення героя на екрані).
13. Якщо R(X1,Y1)=0,то переходимо до пункту 15.
14. Виконуємо привласнення X:=X1,Y:=Y1. Переходимо до пункту 11.
15. Все !!!

```
#include "iostream"
```

```
using namespace std;
```

```
int a[1000][1000];
```

```
int main()

{ int n,m,i,j,k,f;

cin>>n>>m;

for (i=1;i<=n;i++)

for (j=1;j<=m;j++)cin>>a[i][j];

k=2;

f=1;

while (f==1)

{f=0;

for (i=1;i<=n;i++)

for (j=1;j<=m;j++) {

if (a[i][j]==k && a[i-1][j]==1) {f=1;a[i-1][j]=k+1;}
```

```
if (a[i][j]==k && a[i+1][j]==1) {f=1;a[i+1][j]=k+1;}

if (a[i][j]==k && a[i][j-1]==1) {f=1;a[i][j-1]=k+1;}

if (a[i][j]==k && a[i][j+1]==1) { f=1;a[i][j+1]=k+1;}

}

k++;

}

for (i=1;i<=n;i++){

for (j=1;j<=m;j++) cout<<a[i][j]<< " " ;

cout<<endl;}

return 0;

}
```