문제 2

최댓값(L)

<그림 1>과 같이 9×9 격자판에 쓰여진 81개의 자연수가 주어질 때, 이들 중 최댓값을 찾고 그 최댓값이 몇 행 몇 열에 위치한 수인지 구하는 프로그램을 작성하시오.

예를 들어, 다음과 같이 81개의 수가 주어질 경우에는 이들 중 최댓값은 90이고, 이 값은 5행 7열에 위치한다.

	1열	2열	3열	4열	5열	6열	7열	8열	9열
1행	3	23	85	34	17	74	25	52	65
2행	10	7	39	42	88	52	14	72	63
3행	87	42	18	78	53	45	18	84	53
4행	34	28	64	85	12	16	75	36	55
5행	21	77	45	35	28	75	90	76	1
6행	25	87	65	15	28	11	37	28	74
7행	65	27	75	41	7	89	78	64	39
8행	47	47	70	45	23	65	3	41	44
9행	87	13	82	38	31	12	29	29	80

<그림 1>

입력

첫째 줄부터 아홉째 줄까지 한 줄에 아홉 개씩 자연수가 주어진다. 주어지는 자연수는 100보다 작다.

출력

첫째 줄에 최댓값을 출력하고, 둘째 줄에 최댓값이 위치한 행 번호와 열 번호를 빈칸을 사이에 두고 차례로 출력한다. 최댓값이 두 개 이상인 경우 그 중 한 곳의 위치를 출력한다.

입력 예	출력 예
3 23 85 34 17 74 25 52 65 10 7 39 42 88 52 14 72 63 87 42 18 78 53 45 18 84 53 34 28 64 85 12 16 75 36 55 21 77 45 35 28 75 90 76 1 25 87 65 15 28 11 37 28 74 65 27 75 41 7 89 78 64 39 47 47 70 45 23 65 3 41 44 87 13 82 38 31 12 29 29 80	90 5 7

출처: 한국정보올림피아드(2007 지역예선 중고등부)

풀이

이 문제는 2차원 구조를 선형으로 모두 탐색하면 쉽게 해결할 수 있는 문제이다. 2차원 구조는 행 우선으로 탐색하는 방법과 열 우선으로 탐색하는 방법이 있는데, 이 문제는 어떤 방법으로 탐색해도 관계없으며, 일반적으로는 행 우선 탐색을 많이 사용한다.

5행 4열의 2차원 배열	5행 4열의 2차원 배열
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20	1 6 11 16 2 7 12 17 3 8 13 18 4 9 14 19 5 10 15 20
[2차원 구조에서의 행 우선 탐색 순서]	[2차원 구조에서의 열 우선 탐색 순서]

다음은 행 우선을 반복문으로 구현한 소스코드이다.

줄	코드	참고

1
2
3
4
5
6
	for(int row=0; row<5; row++)
	{
	  for(int col=0; col<4; col++)
	  printf("[%d, %d]", row, col);
	  puts("");
	}	

다음은 열 우선을 반복문으로 구현한 소스코드이다.

줄	코드	참고

1
2
3
4
5
6
	for(int col=0; col<4; col++)
	{
	  for(int row=0; row<5 ; row++)
	  printf("[%d, %d]\n", row, col);
	  puts("");
	}	

[0, 0] [0, 1] [0, 2] [0, 3] [1, 0] [1, 1] [1, 2] [1, 3] [2, 0] [2, 1] [2, 2] [2, 3] [3, 0] [3, 1] [3, 2] [3, 3] [4, 0] [4, 1] [4, 2] [4, 3]	[0, 0] [1, 0] [2, 0] [3, 0] [4, 0] [0, 1] [1, 1] [2, 1] [3, 1] [4, 1] [0, 2] [1, 2] [2, 2] [3, 2] [4, 2] [0, 3] [1, 3] [2, 3] [3, 3] [4, 3]
[2차원 구조에서의 행 우선 출력 결과]	[2차원 구조에서의 열 우선 출력 결과]

이제 문제를 해결하는 방법에 대해서 알아보자.

탐색하기 전 먼저 해를 저장할 변수인 ans를 0으로 초기화한다. 여기서 주의할 점은 각 원소들 중 음수값이 존재할 경우 최댓값을 구하기 위해 ans를 0으로 초기화하면 안 된다는 점이다. 이 문제는 음수값이 존재하지 않기 때문에 ans를 0으로 초기화하고 문제를 해결한다.

참고로 어떤 변수에 값을 초기화하는 몇 가지 방법을 소개한다. 일단 int형의 최댓값은 0x7fffffff(2,147,483,647)이며, 최솟값은 0x80000000(−2,147,483,648)이다. 엄밀하게 최대, 최소를 지정할 때 이 값을 이용하면 되며, 16진법을 이용하면 쉽게 처리할 수 있다.

여기서 주의할 점은 위 값들을 설정한 후 값을 증가시키거나 감소시키면 오버플로(overflow)로 인하여 답이 잘못될 수 있다. 예를 들어 다음 명령을 보자.

줄	코드	참고
1	`int max = 0x7fffffff;`	
2	`max = max + 1;`	

위 예의 경우에 max값이 최댓값이었는데, 여기서 1을 증가하면 오버플로가 발생하여 max값은 음수가 된다. 따라서 이런 점을 방지하기 위하여 적어도 2배 정도라 하더라도 오버플로가 발생하지 않도록 처리하는 경우가 많다. 이럴 때는 주로 최댓값을 987654321 등의 자릿수도 쉽게 알 수 있고 2배를 하더라도 정수 범위에 있는 수 등을 활용하는 경우가 많다. 문제에 따라서는 탐색하고자 하는 데이터 중에서 임의의 한 값을 최댓값 또는 최솟값으로 결정하는 방법도 있다.

이러한 점들도 자신만의 코딩 스타일을 구성하는 요소가 되므로 자신만의 방식으로 최대, 최소 등을 정하는 방법을 익혀두자. 위 문제를 해결하는 소스코드는 다음과 같다.

줄	코드	참고
1	`#include <stdio.h>`	
2	`int A[10][10], ans, mi, mj;`	
3	`void input()`	
4	`{`	
5	`  for(int i=0; i<9; i++)`	
6	`    for(int j=0; j<9; j++)`	
7	`      scanf("%d", &A[i][j]);`	
8	`}`	
9		
10	`int solve()`	
11	`{`	
12	`  for(int i=0; i<9; i++)`	
13	`    for(int j=0; j<9 ; j++)`	
14	`      if(ans < A[i][j])`	
15	`      {`	
16	`        ans=A[i][j];`	
17	`        mi=i+1;`	
18	`        mj=j+1;`	
19	`      }`	
20	`}`	
21		
22	`int main()`	
23	`{`	
24	`  input();`	
25	`  solve();`	
26	`  printf("%d\n%d %d\n", ans, mi, mj);`	
27	`  return 0;`	
28	`}`	

가장 일반적으로 해결할 수 있는 방법이고 이 경우 계산량은 O(행$\times$열)이 된다. 이를 보다 효율적으로 바꾸기 위해서, 입력받으면서 바로 처리할 수도 있으며, ans, mi, mj를 모두 쓰지 않고 mi, mj만 가지고 처리하는 방법을 소개한다.

줄	코드	참고
1	#include <stdio.h>	
2		
3	int A[10][10], mi, mj;	
4		
5	void input_solve()	
6	{	
7	for(int i=0; i<9; i++)	
8	for(int j=0; j<9; j++)	
9	{	
10	scanf("%d", &A[i][j]);	
11	if(A[mi][mj]<A[i][j])	
12	mi=i, mj=j;	
13	}	
14	}	
15		
16	int main()	
17	{	
18	input_solve();	
19	printf("%d\n%d %d\n", A[mi][mj], mi+1, mj+1);	
20	return 0;	
21	}	

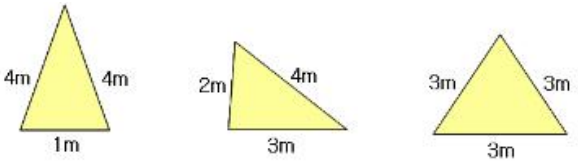
이 방법도 잘 이해하고 익혀두면 코딩의 실수와 시간을 줄일 수 있다.

문제 3

삼각화단 만들기(S)

주어진 화단 둘레의 길이를 이용하여 삼각형 모양의 화단을 만들려고 한다. 이 때 만들어진 삼각형 화단 둘레의 길이는 반드시 주어진 화단 둘레의 길이와 같아야 한다. 또한, 화단 둘레의 길이와 각 변의 길이는 자연수이다. 예를 들어, 만들고자 하는 화단 둘레의 길이가 9m라고 하면,

- 한 변의 길이가 1m, 두 변의 길이가 4m인 화단
- 한 변의 길이가 2m, 다른 변의 길이가 3m, 나머지 변의 길이가 4m인 화단
- 세 변의 길이가 모두 3m인 3가지 경우의 화단을 만들 수 있다.



화단 둘레의 길이를 입력받아서 만들 수 있는 서로 다른 화단의 수를 구하는 프로그램을 작성하시오.

입력

화단의 길이 n이 주어진다.(단, 1 <= n <= 100)

출력

출력내용은 입력받은 n으로 만들 수 있는 서로 다른 화단의 수를 출력한다.

입력 예	출력 예
9	3

출처: 한국정보올림피아드(2002 전국본선 초등부)

풀이

이 문제는 입력구조로 볼 때, 전체탐색으로 해결하려면 선형구조인지 비선형구조인지 등을 판단하기 쉽지 않다. 즉, 지금까지 다루었던 문제들보다 문제를 구조화하는 데에 조금 더 어려움이 있는 문제라고 할 수 있다.

이 문제에서는 세 변의 길이의 합인 n을 알고 있는 상태에서 삼각형의 세 변의 길이를 구하는 문제이므로 각 변을 a, b, c라고 하면 변의 길이 a의 길이를 1부터 n까지 정하고, b, c도 같은 방법으로 순차적으로 정해나가는 방법으로 전체탐색을 할 수 있다. 이렇게 정할 경우 3차원 구조를 가지는 선형 구조가 된다.

여기서 주의할 점은 a, b, c를 각각 1부터 n까지 탐색한다면 각 삼각형이 여러 번 중복되어 구해진다. 예를 들어 화단의 길이가 9일 때, 2, 4, 3으로 골랐다면 3, 4, 2와 4, 2, 3등은 모두 같은 삼각형이지만 따로 카운팅하게 된다. 따라서 처음부터 a를 가장 짧은 변, c를 가장 긴 변으로 정하면 문제 해결이 간단해진다.

그리고 이 문제의 입력 n의 최댓값이 100이므로 100^3으로 접근하더라도 충분히 해결가능하기 때문에 전체탐색법으로 해결해보자.

먼저 3차원 구조로 세변의 길이를 전체 탐색하는 구문을 작성해보자.

줄	코드	참고
1	`int count=0;`	
2	`for(int a=1; a<=n; a++)`	
3	`  for(int b=a; b<=n; b++)`	
4	`    for(int c=b; c<=n; c++)`	
5	`    {`	
6	`      if(count%5 == 0) puts("");`	
7	`        count++;`	
8	`      printf("[%d %d %d]\t", a, b, c);`	
9	`    }`	

위 구조대로 탐색하면 각 변의 길이가 1부터 5까지의 모든 경우에 대해서 조사한다. 단 각 변의 길이를 a, b, c라고 할 때, $a \le b \le c$ 를 만족하는 값들만 탐색한다. 위 탐색의 결과를 출력하면 다음과 같다. 출력할 때, 한 줄에 5개씩만 출력하도록 count변수를 활용

하였으므로 참고한다.

[1 1 1]	[1 1 2]	[1 1 3]	[1 1 4]	[1 1 5]
[1 2 2]	[1 2 3]	[1 2 4]	[1 2 5]	[1 3 3]
[1 3 4]	[1 3 5]	[1 4 4]	[1 4 5]	[1 5 5]
[2 2 2]	[2 2 3]	[2 2 4]	[2 2 5]	[2 3 3]
[2 3 4]	[2 3 5]	[2 4 4]	[2 4 5]	[2 5 5]
[3 3 3]	[3 3 4]	[3 3 5]	[3 4 4]	[3 4 5]
[3 5 5]	[4 4 4]	[4 4 5]	[4 5 5]	[5 5 5]

위와 같이 탐색을 하면 모든 경우에 대해서 탐색한다. 따라서 각 건에 대해서 삼각형 여부만 판단하면 된다. 세 변의 길이로 삼각형을 판단하는 기본 조건은 다음과 같다.

$$a + b > c$$

그리고 이 문제에서만 적용되는 조건이 있다. 세 변의 길이의 합이 n이어야 한다. 따라서 다음 조건도 만족해야 한다.

$$a + b + c = n$$

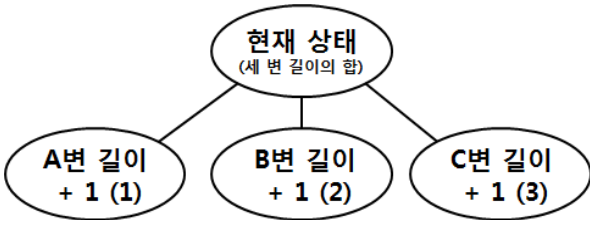
위 탐색방법과 삼각형의 조건을 적용한 소스코드는 다음과 같다.

줄	코드	참고
1	`#include <stdio.h>`	
2		
3	`int n;`	
4		
5	`int solve()`	
6	`{`	
7	`  int cnt = 0;`	
8	`  scanf("%d", &n);`	
9	`  for(int a=1; a<=n; a++)`	
10	`    for(int b=a; b<=n; b++)`	
11	`      for(int c=b; c<=n; c++ )`	
12	`        if(a+b+c == n && a+b>c)`	
13	`          cnt++;`	
14	`  return cnt;`	
15	`}`	
16		
17	`int main()`	
18	`{`	
19	`  printf("%d\n", solve());`	
20	`}`	

9행~13행이 3차원으로 공간을 탐색해 나가는 과정이다. 이러한 문제와 같이 직접적으로 구조화되어 있지 않은 문제도 해결과정에서 구조화해가며 풀어야 하는 문제가 많다.

12행에서는 세 변의 길이의 합 조건과 삼각형을 이루는 조건을 검사한다.

이 문제는 비선형으로 구조화해서 해결할 수도 있다. 문제에서 주어진 조건들을 이용하여 다음과 같은 탐색구조를 설계할 수 있다.



위 트리를 살펴보면 현재 상태(현재 세 변 길이의 합)가 이 n보다 적으면 (1), (2), (3)의 순으로 깊이우선탐색한다. 그러면 다시 (1), (2), (3)이 각각 새로운 현재 상태가 된다. 계속 깊이우선으로 탐색할 수 있는 상태가 된다.

만약 현재 상태가 n이 되면 탐색을 종료하고 삼각형의 조건이 맞는지 확인한다. a, b, c 세 변이 삼각형을 이루는 조건을 만족한다면 가능한 삼각형의 수를 1증가시킨다.

이렇게 모든 상태를 탐색하면 가능한 모든 경우가 만들어진다. 이를 구현한 소스코드는 다음과 같다.

줄	코드	참고
1	`#include<stdio.h>`	6: 모든 변을 다 썼으면 12: a변을 1증가 계속 탐색 13: b변을 1증가 계속 탐색 14: c변을 1증가 계속 탐색
2		
3	`int cnt;`	
4	`void solve(int n, int a, int b, int c)`	
5	`{`	
6	`  if(a+b+c==n)`	
7	`  {`	
8	`    if(a<=b && b<=c && a+b>c)`	
9	`      cnt++;`	

줄	코드	참고
10	return; }	
11		

12	solve(n, a+1, b, c);
	  solve(n, a, b+1, c);
	  solve(n, a, b, c+1);
	}	

13		
14		
15		
16		

17	int main(void)
	{
	  int n;
	  scanf("%d", &n);
	  solve(n, 1, 1, 1);
	  printf("%d\n", cnt);
	}	

18		
19		
20		
21		
22		
23		

하지만 위 소스코드에는 문제점이 있다. 각 변에 1씩 증가시켜가며 탐색해 나가는데 이는 n이 5이고 최종 상태가 1, 2, 2라고 할 때, b변에서 1을 먼저 증가시켜 나온 1, 2, 2와 c변에서 1을 먼저 증가시켜 나온 1, 2, 2를 서로 다른 경우로 카운트한다. 즉, 같은 모양의 삼각형을 여러 번 중복해서 계산하게 된다.

이를 방지하기 위하여 한 번 삼각형으로 카운트 된 a, b, c에 대해서는 chk[a][b][c]배열의 값을 1로 바꾸어 체크해둔다. 이 방법을 이용하여 카운트 하는 것을 방지할 수 있다.

위 방식으로 작성한 소스 프로그램은 아래와 같다.

줄	코드	참고
1	#include<stdio.h>	3: chk는 중복 체크용 7: 모든 변을 다 썼으면 9: 삼각형 조건 만족 12: 중복 방지용 체크 16: a변을 1증가 계속 탐색 17: b변을 1증가 계속 탐색 18: c변을 1증가 계속 탐색
2		
3	int cnt, chk[21][21][21];	
4		
5	void solve(int n, int a, int b, int c)	
6	{	
7	if(a+b+c==n)	
8	{	
9	if(a<=b && b<=c && a+b>c && chk[a][b][c]==0)	
10	{	
11	cnt++;	
12	chk[a][b][c]=1;	
13	}	

줄	코드	참고
14	return; }	
15		

16	solve(n, a+1, b, c);
	  solve(n, a, b+1, c);
	  solve(n, a, b, c+1);
	}	

17		
18		
19		
20		

21	int main(void)
	{
	  int n;
	  scanf("%d", &n);
	  solve(n, 1, 1, 1);
	  printf("%d\n", cnt);
	}	

22		
23		
24		
25		
26		
27		

이와 같이 중복 방지를 위해서 체크하는 방법은 자주 활용되므로 잘 익혀둘 수 있도록 한다. 그리고 만약 n의 크기가 10,000이상의 값이라면 어떻게 해결해야할지 고민해보기 바란다. 이 방법들은 모두 시간이 너무 많이 걸리기 때문에 n값이 커질 경우 제한된 시간 이내에 해를 구할 수 없다.

문제 4

고기잡이(S)

우리나라 최고의 어부 정올이가 이번에 네모네모 배 고기잡이 대회에 참가한다.

이 대회에는 3개의 라운드가 있는데, 첫 번째 라운드는 1차원 형태로 표현될 수 있는 작은 연못에서 길쭉한 그물을 던져서 최대한 많은 고기를 잡는 것이 목적이다.

1라운드의 예를 들면 연못의 크기가 1*6이고 물고기의 위치와 가치가 다음과 같다고 하자.

1 0 2 0 4 3

여기서 그물의 크기는 1*3이라고 할 때, 잡을 수 있는 방법은 (1 0 2), (0 2 0), (2 0 4), (0 4 3)의 4가지 방법이 있다.

이 중 가장 이득을 보는 방법은 마지막 방법 0 + 4 + 3 = 7이다. 따라서 주어진 경우의 최대 이득은 7이 된다. 정올이는 최대한 가치가 큰 물고기를 잡아서 우승하고 싶어 한다.

연못의 폭과 각 칸에 있는 물고기의 가치, 그물의 가로의 길이와 세로의 길이가 주어질 때, 잡을 수 있는 물고기의 최대이득을 구하는 프로그램을 작성하시오.

입력

첫 번째 줄에 연못의 폭 N이 입력된다. (N <= 100 인 자연수)
두 번째 줄에 그물의 폭 W가 입력된다. (W <= N 인 자연수)
세 번째 줄 W개의 물고기의 가치가 공백으로 구분되어 주어진다. 각 물고기의 가치는 7이하의 자연수이다. 0일 경우에는 물고기가 없다는 의미이다.

출력

잡을 수 있는 물고기의 최대 가치를 출력한다.

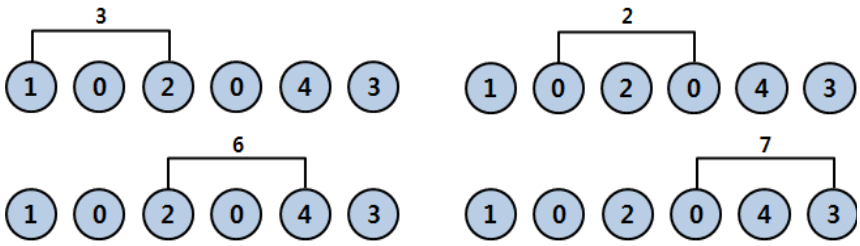
입력 예	출력 예
6 3 1 0 2 0 4 3	7

풀이

이 문제는 전체탐색법을 이용하여 간단하게 해결할 수 있다. 폭이 N인 연못에서 폭이 W인 그물을 던졌을 때 최대 이득을 얻을 수 있는 구간을 찾는 문제이다. 가장 단순한 방법으로 N개의 주어진 수들 중 연속된 W개의 수들을 탐색하여 합을 구한 다음 최댓값을 갱신하는 방법으로 접근할 수 있다.

먼저 첫 번째 데이터부터 탐색하여 W개의 합을 구한 다음 최댓값을 갱신하고, 두 번째 데이터부터 탐색하여 W개의 합을 구하여 최댓값을 갱신한다. 이런 방법으로 모든 구간을 전체탐색법으로 확인할 수 있다.

입출력 예의 경우 다음과 같은 과정으로 해를 구해나간다.



따라서 위의 경우 해는 7이 된다.

위의 과정을 보면 탐색을 시작하는 지점이 0번으로부터 시작하여 1씩 증가하는 것을 알 수 있으며, 시작점을 지정하면 그물의 폭인 W만큼 탐색을 진행한다. 따라서 마지막 탐색의 시작 지점은 $N-W+1$ 이 된다. 핵심 탐색 부분을 구현하면 다음과 같다.

줄	코드	참고
1	`for(int i=0; i<N-W+1; i++)`	
2	`{`	
3	`  for(int j=0; j<W; j++)`	
4	`    printf("%d ", i+j);`	
5	`  puts("");`	
6	`}`	

$n = 8$ 이고 $w = 5$ 일 때, 위 탐색방법의 출력결과는 결과는 다음과 같다.

```
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
```

즉, [0, 4] 구간, [1, 5]구간, [2, 6]구간, [3, 7]구간으로 모두 4번을 검사한다.

위 소스코드에서 $N - W + 1$ 을 생각하기 어려운 경우에는 배열을 좀 더 크게 잡은 후 다음과 같이 작성해도 관계없다.

줄	코드	참고
1	`for(int i=0; i<N; i++)`	
2	`{`	
3	`  for(int j=0; j<W; j++)`	
4	`    printf("%d ", i+j);`	
5	`  puts("");`	
6	`}`	

위와 같이 작성하면 생각하기 쉽기 때문에 빠른 시간에 코딩이 가능하다. 위의 코드의 출력결과는 다음과 같다.

```
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9
6 7 8 9 10
7 8 9 10 11
```

빨간색으로 표시된 부분은 실제로 데이터가 0이 들어있어 구간의 합을 구하더라도 해를 구하는데 영향을 미치지 않는다.

위의 아이디어 들을 이용하여 문제를 해결한 소스코드는 다음과 같다.

줄	코드	참고
1	`#include <stdio.h>`	
2		
3	`int data[101], N, W, ans = 0;`	
4		
5	`int main()`	
6	`{`	
7	`  scanf("%d%d", &N, &W);`	
8	`  for(int i=0; i<N; i++)`	
9	`    scanf("%d", data+i);`	
10		
11	`  for(int i=0; i<N+W-1; i++)`	
12	`  {`	
13	`    int sum = 0;`	
14	`    for(int j=0; j<W; j++)`	
15	`      sum+=data[i+j];`	
16	`    if(sum>ans) ans=sum;`	
17	`  }`	
18		
19	`  printf("%d", ans);`	
20	`}`	

이 알고리즘의 계산량은 1~N의 각 위치에 대해서 W만큼 탐색을 하므로 $O(NW)$가 됨을 알 수 있다.

문제에서 제시한 N의 최대치가 100,000이 입력되고 그물의 크기가 적당히 크면 수행시간이 많이 걸리므로 좀 더 효율적인 알고리즘이 필요하다.

문제 5

고기잡이(L)

우리나라 최고의 어부 정올이가 이번에 네모네모 배 고기잡이 대회에 참가한다.

이 대회에는 3개의 라운드가 있는데, 두 번째 라운드는 2차원 형태로 표현될 수 있는 작은 연못에서 길쭉한 그물을 던져서 최대한 많은 고기를 잡는 것이 목적이다.

1라운드의 예를 들면 연못의 크기가 1*6이고 물고기의 위치와 가치가 다음과 같다고 하자.

1 0 2 0 4 3

여기서 그물의 크기는 1*3이라고 할 때, 잡을 수 있는 방법은 (1 0 2), (0 2 0), (2 0 4), (0 4 3)의 4가지 방법이 있다.

이 중 가장 이득을 보는 방법은 마지막 방법 0 + 4 + 3 = 7이다. 따라서 주어진 경우의 최대 이득은 7이 된다. 정올이는 최대한 가치가 큰 물고기를 잡아서 우승하고 싶어 한다.

연못의 폭과 각 칸에 있는 물고기의 가치, 그물의 가로의 길이가 주어질 때, 잡을 수 있는 물고기의 최대이득을 구하는 프로그램을 작성하시오.

입력

첫 번째 줄에 연못의 폭 N, M이 입력된다. (N, M <= 100 인 자연수)
두 번째 줄에 그물의 폭 W, H가 입력된다. (W <= N, H <= M 인 자연수)
세 번째 줄에 N*M개의 물고기의 가치가 공백으로 구분되어 주어진다. 각 물고기의 가치는 7 이하의 자연수이다. 0일 경우에는 물고기가 없다는 의미이다.

출력

잡을 수 있는 물고기의 최대 가치를 출력한다.

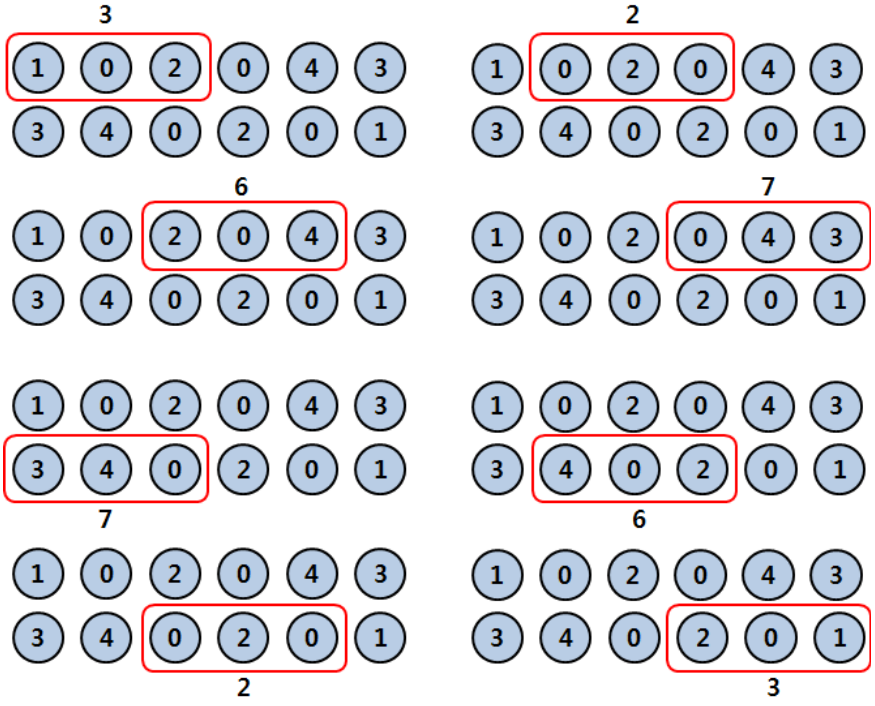
입력 예	출력 예
2 6 1 3 1 0 2 0 4 3 3 4 0 2 0 1	7

풀이

[문제 4] 고기잡이(S)가 2차원으로 확장된 형태의 문제이다. 따라서 이 문제 N, M값이 크지 않으므로 2차원 전체탐색법을 이용하여 해결할 수 있다.

먼저 첫 번째 줄의 데이터부터 탐색하여 $W*H$ 개의 합을 구한 다음 최댓값을 갱신하고, 그 다음 순서대로 탐색하여 $W*H$ 개의 합을 구하여 최댓값을 갱신한다. 이런 방법으로 (1,1)~(N, M) 구간까지 확인한다.

입출력 예시(N=2, M=6, W=1, H=3)를 예로 들면, 다음과 같다.



이 과정을 거쳐 최댓값이 7임을 알 수 있다. 구하는 과정은 앞에서 다룬 것과 대부분 동일하다. 소스코드는 다음과 같다.

줄	코드	참고
1	#include <stdio.h>	
2		
3	int data[100][100];	
4	int N, M, H, W;	
5	int res=0;	
6		
7	int main()	
8	{	
9	scanf("%d %d", &N, &M);	
10	scanf("%d %d", &H, &W);	
11	for(int i=0; i<N; i++)	
12	for(int j=0; j<M; j++)	
13	scanf("%d", &data[i][j]);	
14		
15	for(int i=0; i<N-H+1; i++)	
16	{	
17	for(int j=0; j<M-W+1; j++)	
18	{	
19	int sum = 0;	
20	for(int a=0; a<H; a++)	
21	for(int b=0; b<W; b++)	
22	sum+=data[i+a][j+b];	
23	if(sum>res) res=sum;	
24	}	
25	}	
26	printf("%d", res);	
27	}	

위 두 알고리즘의 계산량은 N*M의 위치에 대해서 H*W만큼 탐색을 하므로 $O(NMHW)$가 됨을 알 수 있다.

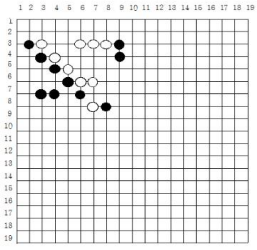
N*M의 최대치가 300,000이고 그물의 크기가 적당히 크면 수행시간이 매우 많이 걸린다.

1차원에서는 전체탐색을 해도 2중 반복문이 되지만 2차원에서는 4중 반복문이 되므로 매우 비효율적인 알고리즘이다. N, M의 값이 조금만 커져도 제한 시간 이내에 해를 구할 수 없다.

문제 6

오목

오목은 바둑판에 검은 바둑알과 흰 바둑알을 교대로 놓아서 겨루는 게임이다. 바둑판에는 가로, 세로 19개의 선으로 이루어져 있다.



오목은 위의 그림에서와 같이 같은 색의 바둑알이 연속적으로 다섯 알이 놓이면 그 색이 이기게 된다. 여기서 연속적이란 가로, 세로 또는 대각선 방향 모두를 뜻한다.

즉, 위의 그림은 검은색이 이긴 경우이다. 하지만 여섯 알 이상이 연속적으로 놓인 경우에는 이긴 것이 아니다. 입력으로 바둑판의 어떤 상태가 주어졌을 때, 검은색이 이겼는지, 흰색이 이겼는지 또는 아직 승부가 결정되지 않았는지를 판단하는 프로그램을 작성하시오.

단, 검은색과 흰색이 동시에 이기거나 검은색 또는 흰색이 두 군데 이상에서 동시에 이기는 경우는 입력으로 들어오지 않는다.

입력

입력 파일은 19줄에 각 줄마다 19개의 숫자로 표현되는데, 검은 바둑알은 1, 흰 바둑알은 2, 알이 놓이지 않은 자리는 0으로 표시되며, 숫자는 한 칸씩 띄어서 표시된다.

출력

첫 번째 줄에 검은색이 이겼을 경우에는 1을, 흰색이 이겼을 경우에는 2를, 아직 승부가 결정되지 않았을 경우에는 0을 출력한다. 검은색 또는 흰색이 이겼을 경우에는 둘째 줄에 연속된 다섯 개의 바둑알 중에서 가장 왼쪽에 있는 바둑알(연속된 다섯 개의 바둑알이 세로로 놓인 경우, 그중 가장 위에 있는 것)의 가로줄 번호와 세로줄 번호를 순서대로 출력한다.

입력 예	출력 예
위 그림과 같은 경우	1 3 2

출처: 한국정보올림피아드(2003 전국본선 초등부)

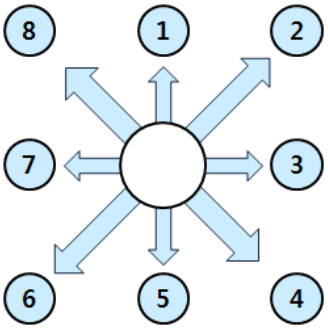
풀이

다양한 경우에 대한 처리를 연습하기에 좋은 문제이다. 우리가 흔히 접할 수 있는 오목 게임에서 흑과 백이 순서를 번갈아 가며 돌을 놓게 되는데 매 순간마다 승패를 검사해야 한다. 이 때 사용되는 알고리즘이 바로 이 문제가 요구하는 것이다. 게임의 규칙은 이미 잘 알고 있으므로 생략하고 어떻게 승패를 검사할 것인지에 대해 고민해 보자.

일단 기본적으로 바둑판을 2차원 배열로 생각하고 2차원 구조로 전체 탐색을 진행하는 것은 당연하다. 전체 탐색을 진행하면서 현재 탐색 중인 돌을 기준으로 오목검사를 행하는 것이 가장 일반적인 아이디어다. 하지만 이 검사에 대해서 생각해야 될 사항들이 많다. 기본적인 탐색 아이디어는 다음과 같다.

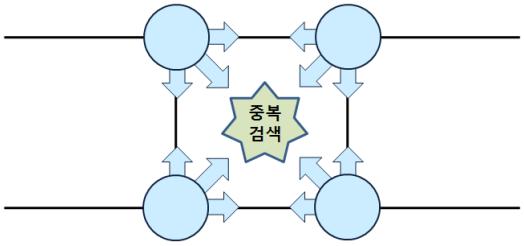
1. (0, 0)부터 (18, 18)까지 행 우선으로 탐색을 시작한다.
2. 현재 탐색 중인 돌을 기준으로 오목이 완성되었는지 검사한다.
3. 완성되지 않았으면 탐색을 진행하기 위해 2번으로 간다.
4. 완성된 돌의 위치와 색깔을 출력한다.

이 문제 해결의 핵심은 2번 항목의 오목이 완성되었는지 검사하는 부분이다. 이 부분은 기본적으로 탐색 중인 돌을 기준으로 아래 그림과 같이 8방향에 대해서 생각해볼 수 있다.



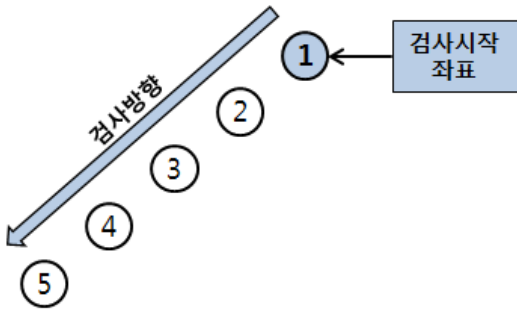
그리고 해당 방향에 돌이 있다면 돌의 개수를 세어야 될 것이다. 하지만 생각한 것보다 다양한 경우가 발생되며, 처리해야할 것들이 많다.

먼저 위 그림과 같은 8방향 검사 알고리즘을 적용하면 다음과 같이 중복으로 검사되는 경우가 발생한다.

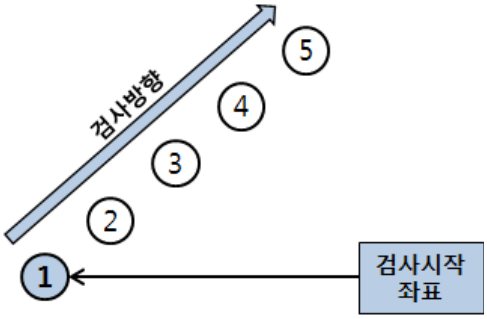


따라서 (0, 0)에서부터 순차적으로 탐색해 나간다면, 8방향 검사를 다 할 필요가 없다. 즉, 검사를 마친 좌표에 대해서는 굳이 검사를 할 필요가 없는 것이다. 그렇다면 8방향 중 몇 곳을 조사해야 할까?

문제에서 요구하는 좌표가 연속된 다섯 개의 바둑알 중에서 가장 왼쪽에 있는 바둑알의 좌표를 요구하므로 검사되는 돌의 좌표에서 오른쪽 방향으로 검사가 이뤄져야 한다.



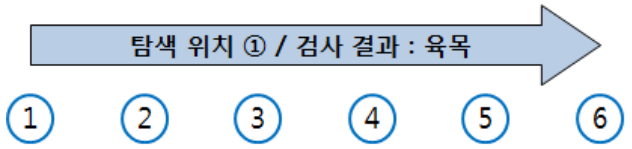
① ↙방향 검사는 검사 시작 돌이 가장 왼쪽 좌표가 될 수 없다.



②↗방향 검사는 검사 시작 돌이 가장 왼쪽 좌표가 될 수 있다.

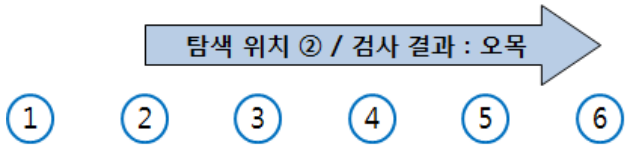
위 그림에서 보는 것과 같이 ↙방향보다는 ↗방향을 선택해야 검사되는 돌이 가장 왼쪽 좌표가 된다. 나머지 방향에 대해서는 별로 애매한 부분이 없을 것이다. 즉, ↓, ↘ , →, ↗ 4방향을 선택하는 것이 현명하다.

이와 같이 4방향을 선택하면 된다. 하지만 또 다른 문제점이 발생할 수 있다. 다음 경우를 확인해 보자.



① 연속되어 같은 돌이 6개 놓인 경우 (육목)이므로 검사결과 오목이 아님

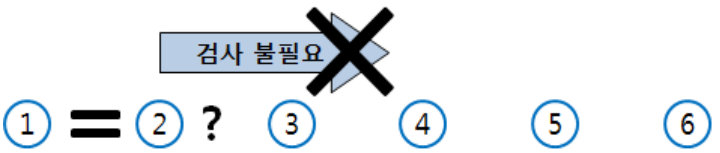
여기까진 괜찮으나 다음 좌표 위치에서는 어떤 결과가 발생할까?



① 2번 돌에서 검사결과 오목임

1번 돌이 있어서 육목인데 ← 방향은 검사하지 않으므로 위와 같이 잘못된 결과가 발생할 수 있다. 따라서 올바른 검사 결과가 나오게 하기 위해서는 ←방향에 같은 돌이 있으면 → 방향은 검사를 하지 말아야 한다.

일반화 시키면 검사 방향의 정반대 방향에 같은 돌이 있으면 검사방향쪽으로 검사를 하지 말아야 하는 것이다. 이전의 검사에서 그 방향은 이미 검사했기 때문이다.



이와 같이 이 문제에서는 다양한 경우에 대해 연습할 수 있는 좋은 문제이다. 하지만 이런 문제는 스스로 소스코드가 맞는지 판단하기가 어려우니 반드시 online judge 등에서 판정을 받아보는 것이 좋다.

전체 소스는 다음과 같다.

줄	코드	참고
1	#include<stdio.h>	2: 19×19, 사방의 끝을 0으로 채움 12: 돌이 놓여있는 경우에 35: 무승부인 경우
2	int a[19+2][19+2];	
3		
4	int main()	
5	{	
6	int i, j;	
7	for(i=1; i<=19; i++)	
8	for(j=1; j<=19; j++)	
9	scanf("%d", &a[i][j]);	
10	for(i=1; i<=19; i++)	
11	for(j=1; j<=19; j++)	
12	if(a[i][j]!=0)	
13	{	
14	if(a[i][j-1]!=a[i][j] && search1(a[i][j], i, j, 1)==1)	
15	{	
16	printf("%d\n%d %d", a[i][j], i, j);	
17	return 0;	
18	}	
19	if(a[i-1][j-1]!=a[i][j] && search2(a[i][j], i, j, 1)==1)	
20	{	
21	printf("%d\n%d %d", a[i][j], i, j);	
22	return 0;	
23	}	
24	if(a[i-1][j]!=a[i][j] && search3(a[i][j], i, j, 1)==1)	
25	{	
26	printf("%d\n%d %d", a[i][j], i, j);	
27	return 0;	
28	}	
29	if(a[i+1][j-1]!=a[i][j] && search4(a[i][j], i, j, 1)==1)	
30	{	
31	printf("%d\n%d %d", a[i][j], i, j);	
32	return 0;	
33	}	
34	}	
35	printf("0");	
36	return 0;	
37	}	

줄	코드	참고
1	int search1(int color, int i, int j, int cnt)	1: → 방향 7: ↘ 방향 14: ↓ 방향 21: ↗ 방향
2	{	
3	for(; color==a[i][j+1]; j++)	
4	cnt++;	
5	return cnt==5 ? 1:0;	
6	}	
7	int search2(int color, int i, int j, int cnt)	
8	{	
9	for(; color==a[i+1][j+1]; i++, j++)	
10	cnt++;	
11	return cnt==5 ? 1:0;	
12	}	
13		
14	int search3(int color, int i, int j, int cnt)	
15	{	
16	for(; color==a[i+1][j]; i++)	
17	cnt++;	
18	return cnt==5 ? 1:0;	
19	}	
20		
21	int search4(int color, int i, int j, int cnt)	
22	{	
23	for(; color==a[i-1][j+1]; i--,j++)	
24	cnt++;	
25	return cnt==5 ? 1:0;	
26	}	