정수 삼각형

1. 문제풀이 방법(알고리즘)

정수 삼각형 문제에서 마지막 레벨까지 내려올 때 합이 가장 큰 경로와 그 때의 합을 구하기 위해서는 배열에 레벨1부터 마지막 레벨까지 내려오면서 각 정수에 도착할 때까지의 합의 최대값을 저장하는 동적계획법을 이용해서 구한다. 최종적으로 마지막 레벨에서 저장한 이 배열의 값이 가장 큰 값을 구하면 최적의 경로의 합이 된다. 또한, 최적의 경로를 구하기 위해서는 각 정수에 도착할 때 배열에 그 이전 지나온 위치를 저장하고, 이 배열을 이용해 최적에 경로에 해당하는 정수 값들을 구한다.

1. 소스코드 & 예제
2. **#include <iostream>**
3. **#include <algorithm>**
4. **using namespace std;**
5. **#define MAX 1000**
6. **int main(void)**
7. **{**
8. **int n=0;**
9. **int arr[MAX][MAX]={};**
10. **int best[MAX][MAX]={};**
11. **//-1 "", 0 "L", 1 "R"**
12. **int path[MAX][MAX]={};**
14. **cin>>n;**
16. **for(int i=0; i<n; i++)**
17. **for(int j=0; j<=i; j++)**
18. **cin>>arr[i][j];**
20. **best[0][0]=arr[0][0];**
21. **path[0][0]=-1;**
22. **for(int i=1; i<n; i++)**
23. **for(int j=0; j<=i; j++)**
24. **{**
25. **if(j-1<0)**
26. **{**
27. **best[i][j]=arr[i][j]+best[i-1][j];**
28. **path[i][j]=1;**
29. **}**
30. **else if(j>i-1)**
31. **{**
32. **best[i][j]=arr[i][j]+best[i-1][j-1];**
33. **path[i][j]=0;**
34. **}**
35. **else**
36. **{**
37. **best[i][j]=arr[i][j]+max(best[i-1][j-1],best[i-1][j]);**
38. **if(best[i-1][j-1]==max(best[i-1][j-1],best[i-1][j]))**
39. **path[i][j]=0;**
40. **else**
41. **path[i][j]=1;**
42. **}**
43. **}**
45. **int max=best[n-1][0];**
46. **int max\_index;**
47. **for(int j=0; j<n; j++)**
48. **if(max<best[n-1][j])**
49. **{**
50. **max=best[n-1][j];**
51. **max\_index=j;**
52. **}**
53. **cout<<"sum: "<<max<<endl;**
55. **int path\_num[MAX];**
56. **int i=n-1;**
57. **int j=max\_index;**
58. **int index=n;**
59. **while(index>0)**
60. **{**
61. **path\_num[--index]=arr[i][j];**
62. **if(path[i][j]==0)**
63. **j--;**
64. **i--;**
65. **}**
66. **cout<<"path: ";**
67. **for(int i=0; i<n; i++)**
68. **cout<<path\_num[i]<<" ";**
69. **return 0;**
70. **}**

예제 실행 결과는 다음과 같다.

입력 값:

5

7

3 8

8 1 0

2 7 4 4

4 5 2 6 5

출력 값:

sum: 30

path: 7 3 8 7 5

1. 소스코드 설명

9~13행에서 선언된 변수의 의미는 다음과 같다.

정수 n: 입력 받은 정수 삼각형의 최대 레벨

배열 arr: 입력 받은 정수 삼각형 값을 저장할 배열

배열 best: best[i][j]는 arr[i][j]까지의 레벨 1부터 지나온 값들이 가장 큰 경우의 값을 저장할 배열

배열 path: path[i][j]는 arr[i][j]까지 오는 최적의 경로에서 arr[i][j]를 기준으로 지나온 바로 전 값이 왼쪽이면 0, 오른쪽이면 1, 레벨1의 값이면 -1을 저장할 배열

n과 arr을 입력 받은 후(15~19행), 배열 best와 path를 구하는 과정(21~44행)은 다음과 같다. (동적계획법으로 배열의 값을 저장한다.)

먼저, 레벨 1의 값 best[0][0]=arr[0][0], path[0][0]=-1으로 저장한다. 그 후에는 레벨 2부터 for문을 돌면서(i←1~n-1), 레벨 i에 존재하는 정수 값(arr[i][j] 이때 j←0~i)을 이용해 best와 path의 값들을 구할 것이다. 레벨 i에서 j번째에 존재하는 정수 값 arr[i][j]은 무조건 best[i][j]에 더해져야 하고, 그 값에 도달하기 위해 바로 전 레벨에서 지나올 수 있는 best값들 중 큰 값도 더해져야 한다. 따라서 best[i][j] = arr[i][j] + max(best[i-1][j-1], best[i-1][j]) 가 된다. 이 때 지나온 경로가 될 수 있는 경우가 왼쪽 밖에 없는 경우(j>i-1)에는 best[i][j] = arr[i][j] + best[i-1][j-1]가 되고, 오른쪽 밖에 없는 경우(j-1<0)에는 best[i][j] = arr[i][j] + best[i-1][j]가 된다. path의 값은 나온 경로가 될 수 있는 경우가 왼쪽 밖에 없는 경우(j>i-1)에는 path[i][j]=0이 되고, 오른쪽 밖에 없는 경우(j-1<0)에는 path[i][j]=1이 된다. 두 가지 경로가 모두 존재할 때는 더 큰 값에 따라 path값을 저장한다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| j  i | 0 | 1 | 2 | 3 | 4 |
| 0 | 7 |  |  |  |  |
| 1 | 10 | 15 |  |  |  |
| 2 | 18 | 16 | 15 |  |  |
| 3 | 20 | 25 | 20 | 19 |  |
| 4 | 24 | 30 | 27 | 26 | 14 |

↑예제를 입력했을 때 배열 best의 값

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| j  i | 0 | 1 | 2 | 3 | 4 |
| 0 | -1 |  |  |  |  |
| 1 | 1 | 0 |  |  |  |
| 2 | 1 | 1 | 0 |  |  |
| 3 | 1 | 0 | 0 | 0 |  |
| 4 | 1 | 1 | 0 | 0 | 0 |

↑예제를 입력했을 때 배열 path의 값

46, 47행에서는 max, max\_index를 선언하였다.

정수 max: 마지막 레벨인 n(i=n-1)에서 가장 큰 best[n-1][j]값을 구하기 위한 변수

정수 max\_index: 이 때의 j값을 구하기 위한 변수

저장된 배열 best와 새롭게 정의한 변수를 이용해 최적의 경로에서의 합을 구하는 과정(48~54행)은 다음과 같다.

for문으로 마지막 레벨의 best[n-1][j]를 돌면서(j←0~n-1), 가장 큰 값을 max에, 그 인덱스 값을 max\_index에 저장한다.

56~59행에서 선언된 변수는 다음과 같은 의미를 가진다.

배열 path\_num: 최종적으로 구하는 최적의 경로에 해당하는 정수들을 저장할 배열

정수 i: 마지막 레벨인 n(i=n-1)에서 -1을 하면서 레벨 1(i=0)까지 인덱스를 나타내기 위한 변수

정수 j: path\_num에 저장하기 위해 max\_index부터 시작해 지나온 위치를 구하기 위한 변수

정수 index: n-1값부터 시작해서 0이 될 때까지 path\_num배열의 인덱스를 나타내기 위한 변수

최적의 경로에 해당하는 배열 path\_num을 구하고 출력하는 과정(60~70행)을 다음과 같다.

index는 n값부터 0이 될 때까지(index←n-1~0) while문을 돌면서, arr[i][j]값을 저장하는데 이 때 i값은 레벨을 하나씩 거슬러올라가므로 1씩 작아지고 j값은 왼쪽일 때는 -1을 해주고 오른쪽일 때는 j값을 유지해주면 path\_num이 구해진다. 최종적으로 구한 path\_num은 for문을 통해 출력한다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| index | i | j | arr[i][j] | path[i][j] | path\_num |
| 5 | 4 | 1 | 5 | 1 | path\_num[4]=5 |
| 4 | 3 | 1 | 7 | 0 | path\_num[3]=7 |
| 3 | 2 | 0 | 8 | 1 | path\_num[2]=8 |
| 2 | 1 | 0 | 3 | 1 | path\_num[1]=3 |
| 1 | 0 | 0 | 7 | -1 | path\_num[0]=7 |

↑예제를 입력했을 때 while문 안에서 path\_num배열이 저장되는 과정