2017 KU Programming Contest

Solution Slide

Div.2 Problem-1 : 고려대학교는 사랑입니다

- `printf("고려대학교");`

- 이게 끝입니다.

Div.2 Problem-2 : 팬들에게 둘러싸인 홍준

- `char str[30];`
- `scanf("%s", str);`
- `printf(":fan::fan::fan:\n");`
- `printf(":fan::%s::fan:\n", str);`
- `printf(":fan::fan::fan:\n");`

- 1번보다는 어려워 보이죠?

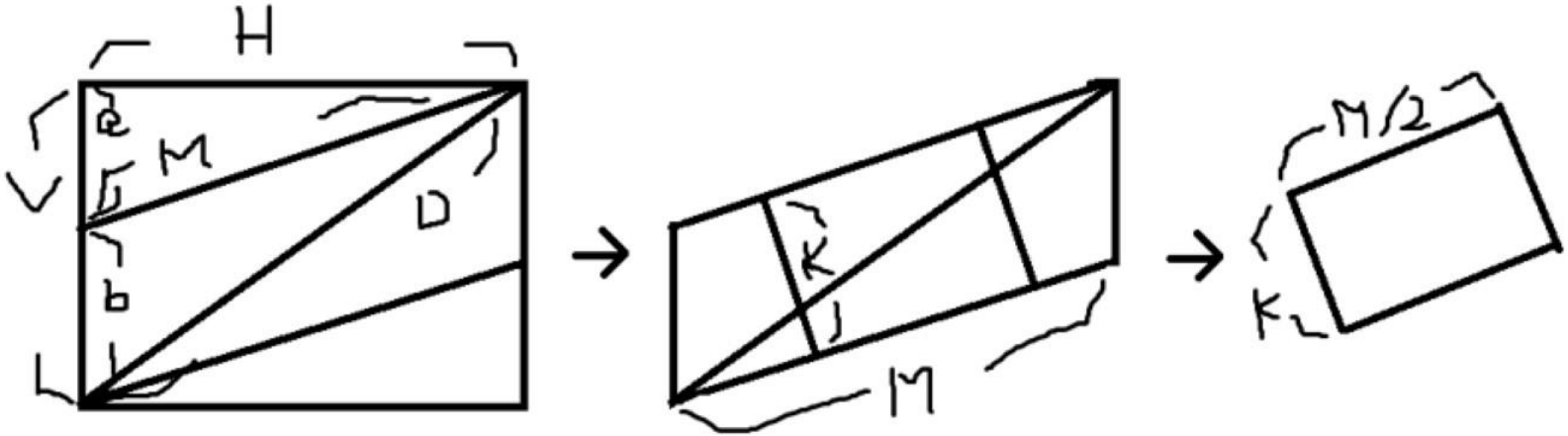
Div.2 Problem-3 : 오늘도 졌다

- 답이 No이려면, 아래 18가지 조건을 모두 만족해야 합니다.
  - 1회 초 득점 <= 0
  - 1회 초 득점 <= 1회 말 실점
  - 1회 초 + 2회 초 <= 1회 말
  - …
  - 1회 초 + 2회 초 + … + 9회 초 <= 1회 말 + 2회 말 + … + 9회 말

- 노가다를 하거나 for문과 배열로 구현하여 해결할 수 있습니다.
- 둘 다 if문은 꼭 써야 합니다.

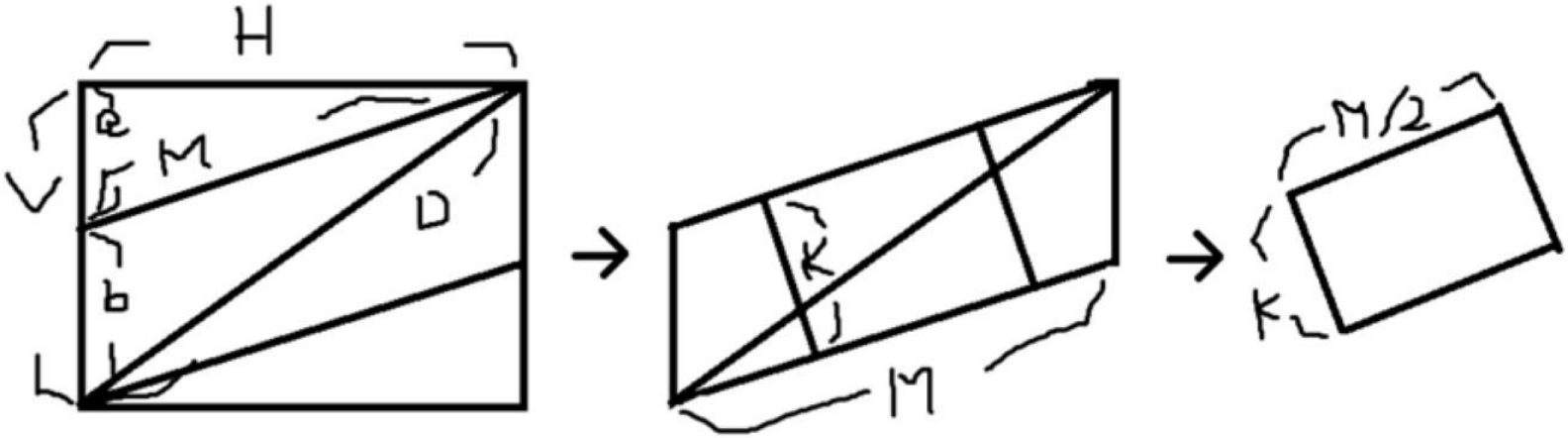
Div.2 Problem-4 : 이음줄

- 접힌 선을 모두 그려봅시다.



Div.2 Problem-4 : 이음줄

- $D^2 = H^2 + V^2$, $M^2 = H^2 + a^2$ (피타고라스 정리)
- a : b = H : D (각의 이등분선 성질)
- M * k = b * H (= 2번째 평행사변형의 넓이)
- 즉, D, a, b, M, k 순으로 값을 구하면 됩니다. 제곱근은 sqrt() 함수를 쓰면 됩니다.



Div.2 Problem-5 & Div.1 Problem-1 : 암호 해독

- 문제의 크기가 작으므로, 가능한 26가지의 비밀 키에 대해서 모두 시도하면 됩니다.

- 먼저, 정해놓은 비밀 키를 이용해서 암호를 해독합니다.
- 그 다음 해독한 문장에 사전에 있는 단어가 있는지 봅니다.
- string.h의 strstr(문장, 단어) 함수를 사용해서 확인하면 됩니다.

Div.2 Problem-6 & Div.1 Problem-2 : 사수빈탕

- 문제 이름이 다소 특이한데, 여기에서 영감을 받았습니다. 저 문제는 이 문제보다 많이 어렵습니다.

- 놀랍게도, 수빈이가 어떻게 움직이더라도 특정 지역까지 가는 시간은 항상 일정합니다.
- 따라서, i번째 사탕바구니에서 사탕을 먹는다면 항상 $\max(0,M-x_i-y_i)$개를 먹습니다.

Div.2 Problem-6 & Div.1 Problem-2 : 사수빈탕

- 좌표의 제한이 300 이하이므로 (300, 300)으로 가면서 먹을 수 있는 최대 사탕의 수를 구하면 됩니다.
  - 동적계획법을 이용하여 문제를 풀면 됩니다.
  - DP[x][y] : (x, y)으로 가면서 먹을 수 있는 최대 사탕의 수
  - 점화식은 D[x][y] = [(x,y)지점에서 먹는 사탕의 수] + max(D[x-1][y], D[x][y-1]) 입니다.

  - x=0 또는 y=0일 때 특수 처리를 하셔야 합니다. 배열에서 -1을 참조하면 런타임 에러가 날 가능성이 큽니다.

Div.1 Problem-3 : 팰린드롬 갯수 구하기

- 동적계획법으로 풀릴 것 같이 생긴 문제입니다. 일단 DP[i][j] = (i~j번째 문자로 이루어진 문자열에서 답) **(※ 편의상 공집합도 답에 포함합니다)**
- 만약 최대 길이를 구하는 문제였다면
  - DMX[i][j] = 1 + DMX[i+1][j-1] (i번째와 j번째 문자가 같을 때)
  - DMX[i][j] = max(DMX[i+1][j], D[i][j+1]) (그 이외의 경우)
- 그렇다고 아무 생각 없이 저 점화식과 비슷하게 짜면 틀립니다.
- 문자열 'abc'에서 'b'가 여러 번 세어집니다.
  - D[1][3] -> D[2][3] -> D[2][2], D[1][3] -> D[1][2] -> D[2][2]

Div.1 Problem-3 : 팰린드롬 갯수 구하기

- 중복을 없애려면, 한 문자를 선택했을 때 그와 대응되는 문자도 같이 선택해야 합니다.
- DP[i][j] = A + B 인데, A는 i번째 문자를 선택하는 경우의 수, B는 그렇지 않은 경우의 수입니다.
- B = DP[i+1][j].
- A = sum(DP[i+1][k-1]) when i번째 문자 == k번째 문자 and $i \le k \le j$

- 이렇게 하면 시간복잡도가 $O(N^3)$이라 small 문제만 풀 수 있습니다.

Div.1 Problem-3 : 팰린드롬 갯수 구하기

- k를 선택하는 노고를 없애기 위해 앞서 언급한 A를 직접 정의하겠습니다.
- DP2[i][j] = (i~j번째 문자로 이루어진 문자열에서 i번 문자를 무조건 선택해야 할 때 답)
- DP[i][j]는 DP[i+1][j] + DP2[i][j]입니다.
- DP2[i][j]는 상황에 따라 점화식이 달라집니다.
    - i번째 문자 != j번째 문자 이면, DP2[i][j] = DP2[i][j-1].
    - 그 이외의 상황에서는 DP2[i][j] = DP2[i][j-1] + DP[i+1][j-1].
- 이렇게 하면 시간복잡도가 $O(N^2)$이 되어 large 문제도 풀 수 있습니다.
- i>j일 때는 DP[i][j]=1, DP2[i][j]=0입니다. 맨 처음에 공집합을 포함한다고 말했는데, 저 1은 공집합을 의미합니다.

Div.1 Problem-4 : 도미노

This problem could be solved with dynamic programming.

But first we need to calculate some additional information about the sticks.

Let's sort them all by positions in ascending order.

If we push ***i-th*** stick to the left it may cause some neighboring sticks also fall to the left.

Let ***leftmostLyingPos(i)*** be the leftmost such stick.

In a similar way let ***rightmostLyingPos(i)*** be the rightmost such stick if we push ***i-th*** stick to the right.

Div.1 Problem-4 : 도미노

Let ***dp(i)*** be the minimal number of pushes needed to make fall all the sticks in positions ***0...i*** inclusive.

We need to consider two options and select one with the minimal value:

1) Make i-th stick fall to the left

In this case we just need to push ***i-th*** stick to the left.

And it'll take ***1+dp(leftmostLyingPos(i)-1)*** pushes in total.

Div.1 Problem-4 : 도미노

2) Make i-th stick fall to the right

Let's consider all such sticks that if we push one to the right it will make ***i-th*** stick also fall to the right. Suppose current such stick is ***j-th***.

Then it will take ***dp(j-1)+1*** pushes to make all ii sticks fall.

So we need to select such stick that this value is minimal.

Div.1 Problem-4 : 도미노

In the second option, when we try to make ***i-th*** stick fall to the right, we don't actually need to consider each stick ***j*** all the time.

Let's just store all these sticks and keep them sorted by ***dp(j-1)+1*** values.

We're only interested in a stick with the minimal such value.

It can be done with Priority Queue and cost ***O(log N)*** for each query.

Also do not forget to delete from this storage all such sticks ***j*** that ***rightmostLyingPos(j)<i***.

Total time complexity is ***O(N log N)***.

Div.1 Problem-5 : Line Friends

Small 문제는 비교적 간단한 방법으로 풀 수 있습니다.

- 먼저 친구 관계를 가중치 1인 간선으로 생각해서 그래프를 만듭니다.
- 그런 후 입맛에 맞는 최단거리 알고리즘을 사용하면 됩니다.
    - BFS / DFS 사용 : O(N^2 Q)
    - Floyd-Warshall 알고리즘 사용 : O(N^3)
    - Dijkstra 알고리즘 사용 : O(N^2 Q)

Div.1 Problem-5 : Line Friends

Large 문제는 선분의 독특한 특징을 이용해야 합니다.

임의의 선분과 임의의 K에 대해서, 그 선분과 K 이하 정도로 친한 선분들의 합집합(편의상 f(그 선분, K)라 하자)은 또 다른 하나의 선분이 됩니다.

- 증명은 수학적 귀납법으로 할 수 있습니다.
- '한 선분'과 '다른 선분'이 K 이하 정도로 친하려면 f('한 선분', K-1)과 '다른 선분'이 만나야 합니다.

Div.1 Problem-5 : Line Friends

- 선분 A와 선분 B의 친한 정도를 구해봅시다.
- 선분 A와 선분 B가 정확히 친한 정도가 K이려면, 선분 B는 f(선분 A, K-1)와 만나야 하며, f(선분 A, K-2)와는 만나면 안 된다.

- 포함 관계(선분 B가 f(선분 A, K)에 포함되어야 한다)로 문제를 풀면 반례가 있습니다.

Div.1 Problem-5 : Line Friends

답을 구하려면, 선분 B와 f(선분 A, K-1)가 만나는 최소 K를 구하면 됩니다.

- 이런 K는 Parametric Search로 구하면 빠르게 구할 수 있습니다.

f(선분 A, K-1)는 어떻게 빠르게 구할까요?

- f(선분 A, x)의 왼쪽 끝 점을 L, 오른쪽 끝 점을 R이라고 하면
- f(선분 A, x+y)의 왼쪽 끝 점은 f(점 L, y)의 왼쪽 끝 점.
- f(선분 A, x+y)의 오른쪽 끝 점은 f(점 R, y)의 오른쪽 끝 점.

Div.1 Problem-5 : Line Friends

이 사실을 이용하면, 2N개의 끝점에 대해서 f(점, 2^i) ($0 \le i \le \log_2 N$) 만 구해놓으면 됩니다.

- 전처리 시간복잡도 : O(N logN)
- f(선분 A, K-1)을 구하는 시간복잡도 : O(logN)

f(점, 1)은 어떻게 빠르게 구할까요?

- f(점, 1)의 왼쪽 끝점은, 점을 지나는 N개의 선분의 끝점 중 가장 왼쪽 점.
- 선분을 오른쪽 끝점 좌표가 큰 순으로 정렬한 후, 오른쪽 끝점부터 차례대로 답을 구하면 됩니다.
- f(점, 1)의 오른쪽 끝점은, 반대 방향으로 구하면 됩니다.

Div.1 Problem-5 : Line Friends

- 설명한 방법대로 차근차근 따르면 $O(N \log N + Q \log^2 N)$ 입니다.

- Parametric Search 과정과 f(선분 A, K-1)을 구하는 과정을 병렬로 처리해서 $O(N \log N + Q \log N)$ 으로 줄일 수 있습니다.

Div.1 Problem-6 : KUBC League

Small 문제는 외판원 순회 문제와 비슷합니다.

- 이 문제를 푸는 알고리즘은 크게 Branch & Bound 방법과 동적계획법 방법이 있습니다.
  - Branch & Bound 알고리즘은 답의 상한을 구해서 필요 없는 경로를 가지치기하는 방법입니다.
  - 동적계획법 방법은 DP[방문한 노드 정보(N-bit)][마지막 방문 노드 번호] 형태의 정의를 세워서 풀면 됩니다.
  - N이 최대 20이므로 시간이 빡빡할 수 있습니다.

Div.1 Problem-6 : KUBC League

Large 문제는 풀리그라는 점을 이용하여 다항 시간으로 푸는 방법입니다.

- A가 B를 이겼다면 A->B 꼴로 간선을 만듭니다.
- 그러면 1번 노드에서 갈 수 없는 노드는 어떻게도 답에 포함시킬 수 없습니다.

- 즉, 1번 노드에서 갈 수 있는 노드가 K개라면, 정답의 크기는 K 이하겠죠.
- 그러니, 크기가 K인 정답을 하나 찾으면 크기가 K+1 이상인 정답은 찾을 필요가 없어요.

Div.1 Problem-6 : KUBC League

- 이제부터는, 순수 아이디어 싸움입니다. 아이디어를 잘 떠올리면 이 문제를 풀 수 있습니다.
- 힌트부터 말하자면, 크기가 K인 정답은 항상 있습니다.
- 그래도 모르시겠다면, 심호흡을 한 번 하고 다음 슬라이드를 보시기 바랍니다. 이 방법은 답을 구하는 여러 방법 중 하나이니, 다른 방법으로 푸셔도 됩니다.

Div.1 Problem-6 : KUBC League

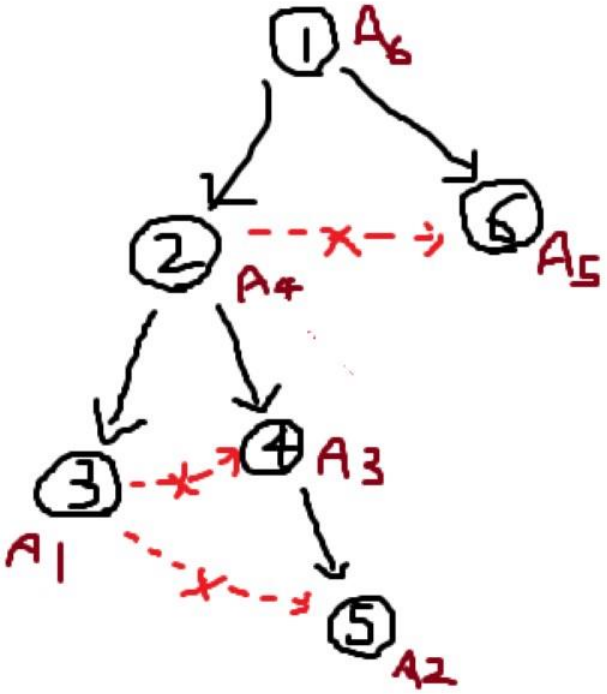
- 1번 노드에서 시작하여 깊이우선탐색(DFS)을 합시다.
- 백트랙(더 나아갈 노드가 없어서 이전 노드로 돌아가는 행위)을 하는 순서대로 A_1, A_2, A_3, …, A_K의 번호를 붙입니다.
- 1번 노드는 항상 가장 끝 번호를 가져갑니다.

- 끝 번호부터 순서대로 노드를 나열하면……

놀랍게도 답이 됩니다!

Div.1 Problem-6 : KUBC League

- DFS를 수행하는 과정을 그림으로 그려봅시다.
- 검은색 화살표는 직접 탐색한 관계를 의미합니다.
- 부모 노드는 항상 자식 노드보다 인덱스가 큽니다.
- A_1번 노드가 A_2번 노드를 이겼다면, A_1에서 A_2로 직접 탐색을 했을 것입니다. 즉, 모순이 생긴다는 의미죠.
- i<j이고 A_i->A_j 이려면 반드시 A_j가 A_i의 조상 노드여야 합니다. A_i 입장에선 이미 방문한 노드 중 인덱스가 더 큰 노드는 조상 노드밖에 없으니까요.



Div.1 Problem-6 : KUBC League

- 역시나, A_i->A_{i+1} 이려면 반드시 A_{i+1}이 A_i의 조상 노드여야 합니다. 두 노드 사이에 다른 노드 A_k가 있다면 $i < k < i+1$이어야 하므로 모순이 생깁니다.
- 만약 A_{i+1}이 A_i의 부모라면 이미 검은 간선 A_{i+1}->A_i이 있으므로 모순입니다.
- 즉, 어떤 일이 있더라도 A_i<-A_{i+1} 입니다.
- 그러니, A_1<-A_2<-A_3<-…<-A_K (= 노드 1) 입니다.