Iamcoder 33rd Open Contest - Solution

2014.11.06 구재현 (gs14004)

(A) 초딩 2 (choding2)

박승원 (gs14041) / 구현, 동적 계획법

1. 100% 풀이 / O(N * M * max(T) * K)

먼저, -n <= x <= n, -m <= y <= m 과 같은 좌표는 배열을 통해서 관리하기 힘드므로, 모든 입출력 좌표에 n,m을 더해서 (0 <= x + n <= 2n , 0 <= y + m <= 2m), 좌표를 쉽게 관리할 수 있도록 한다.

이후에 우리가 구현해야 할 것은 다음과 같다.

* 1. 현재 승원이의 위치를 기반으로, 승원이가 새 좌표에 있을 경우의 수를 센다.

* 2. 폭탄이 터지는 위치에서 승원이가 존재할 경우의 수를 0으로 설정한다.

승원이가 t 시간에 (i,j) 위치에 있을 가능성을 p(t,i,j) 라 할 경우에, p(t+1,i,j) = p(t,i,j) + p(t,i-1,j) + p(t,i+1,j) + p(t,i,j-1) + p(t,i,j+1) 이 성립한다. 이 식을 통해서 t초 뒤에 승원이의 위치를 O(t * n * m) 시간에 계산할 수 있다. 이후 입력으로 들어온 폭탄에 대해서, 폭탄의 폭발 영역을 0으로 설정해 주면 된다.

초기 조건은 물론, p(0,a+n,b+m) = 1이다.

입력값이 상당히 작은 편이므로, 이러한 방법을 통해 충분히 시간 내에 풀 수 있다.

* 32비트 정수형 변수 내의 입력값을 만들어야 하기 때문에, 이 문제는 특별히 10개의 테스트 케이스가 모두 수작업으로 제작되었으며, 풀사이즈 데이터 역시 없습니다.

* 이상하게 어려웠다는 제보가 많았음..

* 승원이 기여어

(B) 공공칠빵 (game)

황동욱 (gs14129) / 구현, 동적 계획법

1. 60% 풀이 / O(N * T)

먼저, 어떠한 사람이 지목당한 횟수를 세는 배열 picked[]를 잡을 경우, 어떠한 사람 t를 지목하는 과정은 단순히 picked[t]++; 를 하는 것으로 O(1)에 처리가능하다.

이제, 그 사람의 주변에 있는 사람을 지목하는 과정을 구현해보자. 원형으로 앉아 있기 때문에, 주기성을 고려하여 x - N번째 = x번째 = x + N 번째라고 감안한다.

* 2 * picked[t] + 1 > N일 경우, t를 제외한 1 ~ N 구간의 모든 사람들은 으악!! 을 외친다.
* 그 외의 경우, (t - picked[t] ~ t-1) 구간과, (t+1 ~ t + picked[t]) 구간에 있는 모든 사람들은 으악!!을 외친다.

원 구간을 적절히 처리한 후 으악을 외친 횟수를 세면, 한번의 지목마다 O(N)의 시간만큼 처리할 수 있다.

2. 100% 풀이 / O (N + T)

먼저, N = 100일때 원을 한바퀴 돌아오는 쿼리 , 이를테면 (95 ~ 3) 과 같은 쿼리는 (95 ~ 100), (1 ~ 3)과 같은 쿼리로 분할해서 처리할 수 있다. 이러한 분할을 거치면 원을 한바퀴 돌아오는 쿼리가 존재하지 않게 문제를 단순화할 수 있다.

으악을 외친 횟수를 세는 배열을 shout[] 라 할때, shout_diff[] 라는 배열을 선언하자. 이 배열은 다음과 같은 값을 저장한다.

shout_diff[x] = shout[x] - shout[x-1]

빈 shout[] 배열에 (3 ~ 8)까지 1을 더한다면,

0 0 1 1 1 1 1 1 0 0...

과 같은 결과가 저장될 것이다. 이제 이 배열의 shout_diff[] 값을 구하면

0 0 1 0 0 0 0 0 -1 0 ..

즉, shout_diff[3]에 1을 더하고, shout_diff[9]에 1을 빼는 과정으로 쿼리를 저장할 수 있다. 이는 한번의 지목을 O(1)만에 처리할 수 있게 해준다.

이러한 방식으로 shout_diff[]를 처음에 저장하고, 이 배열에서 shout[] 배열을 O(N) 만에 유도할 수 있다. 각 쿼리를 처리하는데 O(1), shout[] 배열 유도에 O(N)이 걸리므로, 시간 복잡도는 O(N + T)이다.

* 변화값을 토대로 배열의 값을 불러오는 기술을 사용하는 문제가 이번 대회에 세 문제나 있는데 (..) 모두 동적 계획법으로 분류하려고 합니다.

(C) 사랑의 메시지 (love)

안형서 (gs14061) / 수학, 동적 계획법

1. 100% 풀이 / O(KlgK)

지금 현재 내가 입력한 하트의 개수가 x개라면, 두가지 방법을 통해서 하트의 개수를 늘릴 수 있다.
(Cost(x) = x개를 입력하기 위해 드는 최소 비용으로, Cost(1) = a이다.)

* 1. 하트를 한번 더 입력한다. 하트의 개수는 x개가 될 것이며, 이 때의 비용은 Cost(x) + a일 것이다.

* 2. 하트를 복사한 후 붙여넣는다. 하트의 개수는 n * x (n > 1, n * x <= k) 개가 될 것이며, 이 때의 비용은 Cost(x) + b + c * (n - 1) 이다.

복사 -> 하트 입력 -> 붙여넣기는, 복사 -> 붙여넣기 -> 하트 입력과 똑같기 때문에 이 방법을 통해 모든 상태를 나타낼 수 있다.

이 점화식을 토대로 동적계획법을 돌리면 되며, 시간 복잡도는 O(k / 1 + k / 2 .... k / k) 일 것이다. 간단한 적분을 통해서 (S (dx / x) = lnx), 이것의 상한이 O(klgk) 라는 것을 증명할 수 있다.

반대 방향으로 생각해서, 현재 하트의 개수의 약수를 모두 나열하는 방식으로 동적 계획법을 사용할 수도 있다. 약수를 O(Sqrt(K)) 개 만에 열거하는 방법이 널리 알려져 있기 때문에, 이 방법은 O(K^1.5)에 작동한다. 이 방법 역시 만점을 받을 수 있지만, 재귀 호출을 사용한다면 stack overflow가 발생할 수 있으니, 반복문을 사용할 것을 권한다.

(D) 지하철 2호선 (line2)

구재현 (gs14004) / 동적 계획법

1. 20% 풀이 / O(N(N+M))

D[i] = (1 <= i <= N) 번째 역을 이용하는 측근의 수라 할 때, 역 i를 선택했을 경우 측근의 만족도는 A * D[i] - B * (M - D[i]) 이고, 선택하지 않았을 때 측근의 만족도는 - B * D[i]이다. O(N * M) 의 연산으로 D[i]를 설정할 수 있으며, O(N^2) 의 연산으로 각 구간을 만족하는 측근의 수를 구할 수 있다.

2. 40% 풀이 / O(N^2 + N + M))

B번 문제 공공칠빵과 같이 변화값을 저장하는 배열을 사용하면, D[i]를 O(N+M)으로 계산할 수 있다. 만약에 측근이 만족하는 구간이 (Si > Ei) 일 경우에는 Si ~ N, 1 ~ Ei인 두 측근이라고 정의했을 경우, 역시 쉽게 계산할 수 있다.

3. 100% 풀이 / O(N + M)

먼저. 최적 구간이 Si <= Ei라고 가정하자. 이 문제에서 우리가 할 수 있는 행동은 한 역을 선택하거나 선택하지 않는 두가지 행동 뿐이기 때문에, 먼저 모든 역을 선택하지 않았다고 가정한 후, (선택했을 경우의 만족도 - 선택하지 않았을 때의 만족도) 의 연속된 최대합을 구하는 문제로 바꿀 수 있다.

한 배열의 연속된 최대합은 O(N)에 구할 수 있는 알고리즘이 알려져 있다. (구글에 Maximum subarray problem을 검색하면 자세한 정보를 알 수 있으며, 이 풀이집에는 설명을 생략한다.) Sum(1<=i<=n) (- B * D[i])에, (선택했을 경우의 만족도 - 선택하지 않았을 때의 만족도) 의 연속된 최대합을 더하면, 최적 구간이 Si <= Ei일 때의 답을 도출할 수 있다.

최적 구간이 Si > Ei인 경우에는, 반대로 모든 경우를 선택 했다고 가정한 후, 1 ~ N 구간에서 최소의 만족도를 가지는 부분 배열을 제거 (즉, 선택하지 않음) 한다고 생각하면 문제를 풀 수 있다. 최소 부분배열 문제는 최대 부분배열 문제의 부호를 반전시킨 후 계산하면 해결할 수 있다.

이 방법을 통해서, 기존에 O(N^2)에 계산하던 최대 만족도를 O(N)에 계산할 수 있게 되며, O(N+M)의 알고리즘을 통해 만점을 획득할 수 있다.

(E) 양학군의 전파 계획 (pony)

오선재 (ggoh / gs14067) / 구현, 동적 계획법

1. 10% 풀이 / O(N^2 * M)

소리의 크기를 저장하는 2차원 배열을 잡고, 배열에서의 현재 위치를 (x,y)라 할때, 각 위치에 Di * max(0,Si / Di * max(|Xi - x|,|Yi - y|) 를 더해주면 된다.

2. 40% 풀이 / O(N * (M+N))

우리가 덮어씌우는 앰프 소리의 크기는, 가로로 놓여있는 2N개의 직선, 세로로 놓여있는 2N개의 직선으로도 해석할 수 있다.

역시 B번 공공칠빵과 비슷한 아이디어인데, Dx[i][j] = A[i][j] - A[i][j-1], Dy[i][j] = B[i][j] - B[i-1][j]라고 두자. 가로 세로로 2N개의 직선이 있기 때문에, 한 쿼리당 직선을 모두 칠하는 데 O(N)의 시간이 걸리며, Dx[i][j]와 Dy[i][j] 값에서부터 A[i][j], B[i][j]를 유도하는데 O(N^2) 시간이 걸리며, A[i][j] + B[i][j] >= K인지를 판별하는 데도 그만큼의 시간이 걸리기 때문에, O((M+N) * N) 시간에 문제를 해결할 수 있다.

3. 100% 풀이 / O(M + N^2)

Dx[i][j] 와 Dy[i][j]에 변화값을 저장하는 과정은, 어찌보면 (0, S[i] / E[i], 2 * S[i] / E[i]...) 로 변화하는 대각선 직선 4개를 긋는 것과 마찬가지이다.

Dx[i][j]와 Dy[i][j]의 대각선 변화값을 저장하는 배열 Expy[i][j], Exmy[i][j]를 만든 후, 여기다(+S[i]/E[i], +S[i]/E[i], .. -S[i]) 를 저장해 나가면, 결과적으로 (0, S[i] / E[i] , 2 * S[i] / E[i]... 와 같은 직선을 만들 수가 있다.

이 역시 가중치가 일정한 직선이기 때문에, 대각선 변화값을 저장하는 배열의 변화값을 저장하는 배열 Fxpy[i][j], Fxmy[i][j]를 만든 후, 여기다가 (+S[i]/E[i], 0 ..... -S[i] / E[i]) 값을 저장하고, Expy / Exmy 배열에 -S[i]를 저장하면 쿼리를 O(1)에 처리할 수 있고, Fxmy , Exmy ... 순서대로 A, B 배열을 만드는데 O(N^2) 시간이 들기 때문에, 이 문제를 O(M + N^2)에 풀 수 있다.

* 풀이 적다 빡침.. koistudy에서 Nails 풀어보면 무슨 뜻인지 대강 알거에요.

(F) 홈즈와 뤼팽의 대결 (holmes)

유재민 (gs14069) / 그래프

1. 0% 풀이 / O(N^2 * M)

이 문제를 그래프로 모델링 해보면, 1, N 정점을 제외한 어떠한 1 / 2개의 정점을 비활성화 시켰을 때, 1 ~ N까지의 경로가 존재하냐는 문제로 환원할 수가 있다. O(N^2)의 쌍을 모두 시도해서, DFS / BFS로 N번째 정점으로의 도달 가능 여부를 조사하면 O(N^2(N+M))에 문제를 풀 수 있다.

2. 100% 풀이 / O(N * (N+M))

그래프에서 한 정점이 제거되었을 때, 컴포넌트의 개수가 증가되는 점들을 절점 (cut vertex)라고 한다. Robert Tarjan은 한번의 DFS 만으로 절점을 구하는 알고리즘을 제안했으며, 이 알고리즘을 변형하면 이 문제를 풀 수 있다. (cut vertex를 구하는 알고리즘 역시 설명을 생략한다.)

절점을 구하는 함수를 수정해, pair<int,bool> 값을 반환하는 함수를 만들자. pair의 first 값은 back number를 반환하고, second 값은 이 함수가 n 점에 도달했는 지를 나타낸다.

이 때, x의 DFS Subtree 중, 그의 back number가 x의 방문 순서보다 크거나 같으며, n 점에 도달할 수 있는 점이라면, 이 점은 뤼팽이 가는 경로를 막을 수 있는 경로라는 것을 알 수 있다. 이 DFS의 시간 복잡도는 O(N+M)이다.

왓슨의 경우는 2 ~ N-1까지 모든 점을 막아보면서 경로가 사라졌는지를 검사하면 구할 수 있으며, 이 방법을 통해 시간 복잡도 O(N * (N+M)) 에 문제를 풀 수 있다.

* ~~30% 풀이 = puts(“Lupin win!”);~~

* 문제 Description과 다르게, 1번부터 N번까지 경로가 없는 TC가 2개 있는 사고가 생겼습니다. 준비 미숙에 먼저 사과드리며 koistudy에 올라갈 때는 해당 TC가 제거될 예정입니다.

* 해당 문제를 더 빠르게 풀수 있는 방법이 있다면 알려주세요!

(G) 학습지 뒤섞기 (flip)

이상헌 (gs14080) / 구현, 수학, 그래프

1. 60% 풀이 / O(NlgN)

먼저, 원래 있던 학습지가 한번의 이동 이후 어느 위치로 이동하는 지를 계산해 보자. 첫번째 학습지를 무시하고, 두번째 학습지를 새 위치에 두고, 세번째는 무시, 네번째는 세 위치에.. 두는 과정을 배열을 사용해서 O(N^2)에 구할 수 있다. 하지만, STL의 list 컨테이너를 사용해서 모든 학습지를 리스트에 push하자. iterator를 증가시키면서 매 두번째 학습지들은 새 위치에 둔 후 list에서 지우면, 이를 O(N)에 해결할 수 있다. 이러한 풀이는 조세퍼스 문제 (Josephus Problem)을 푸는 방법으로도 유명하다.

학습지의 1번 이동시의 위치를 구했다면, 이것은 정의역과 치역이 1 ~ N인 전단사함수로 볼 수 있다. 이때, 다음과 같은 함수를 생각해 보자. (이 함수는 실제로 생성되는 함수와 아무 관계가 없다.)

x = 1 2 3 4 5 6 7

y = 2 3 4 1 6 7 5

이때, f^4(1) = 1임을 알 수 있으며, 같은 이유로 f^4(2) = 2, f^4(3) = 3, f^4(4) = 4이다. 즉 1,2,3,4는 4의 주기로 원래 위치로 돌아온다. 같은 방법으로 5,6,7은 주기 3마다 원래 위치로 돌아오며, 이 때 이 함수의 주기는 lcm(4,3) = 12이다. 즉, 모든 함숫값들의 사이클 길이를 모은 후, 그 최소공배수를 출력해주면 된다. 이는 다음과 같은 함수 get_cycle로 구현할 수 있다.

```
get_cycle(x) :

if(visited[x]) return 0;

visited[x] = 1;

return get_cycle(f[x]) + 1;
```

이렇게 각 사이클의 길이를 배열에 집어넣고, 각각의 최소공배수를 구하면 되지만, 이 숫자들은 모두 long long 범위를 초과한다. 때문에, 유클리드 알고리즘을 사용하지 않고, 손으로 구하는 방식으로 최소공배수를 구하면 된다. fact[i] = 최소 공배수에서 소인수 i의 개수라고 할때, get_cycle()의 반환값을 소인수분해하면서 fact[] 배열을 수정할 수 있다. 이후 fact[] 배열의 모든 원소를 모두 곱해가면서 최소공배수를 구하면 된다.

일반적으로 소인수 분해를 하는 알고리즘은 O(Sqrt(N))에 작동한다. get_cycle 함수의 반환값을 모두 cycle 배열에 저장했다면, Sum(Sqrt(cycle[i])) <= Sum(cycle[i]) = N 이기 때문에, 소인수 분해에 드는 총 시간은 O(N)임을 보일 수 있다. 또한, fact[i]는 lgN을 넘지 않음 역시 보일 수 있기에, 시간 복잡도는 O(NlgN)이다. (repeated squaring을 통해서 NlglgN 시간 복잡도를 유도할 수 있지만, 생략한다.)

2. 100% 풀이 / O(NlgN)

get_cycle은 사이클의 길이가 긴 경우 stack overflow를 유발한다. 때문에, 이를 반복문으로 대체하면 100점을 얻을 수 있다.

* 푸신 분들이 더 잘 아시겠지만 이번 대회에서 가장 신선하고 재미있는 문제였습니다 :)
* 학습지의 이동을 O(N^2)에 구하는 알고리즘을 사용해도 100점이 뜨는 걸로 알고 있습니다.