**문제 1. 동전 교환문제 Dynamic Programming Algorithm**

1. 유사코드

* 입력: M(만들고 싶은 금액 e.g. 11), C(동전 종류 배열 e.g. 1, 3, 5)
* 출력: coinUsed(사용한 동전들), numCoin(사용한 최소 동전 개수)

1. dp[0] ← 0 // 0원은 동전 0개로 만들 수 있으니 설정해주기

2. dp[1] ~ dp[M] ← 아주 큰 수 (INF)로 초기화

3. used[0] ~ used[M] ← -1로 초기화 (아직 아무 동전도 안 썼다는 의미)

4. for coin in C: // 모든 동전 종류에 대해

for i from coin to M: // coin부터 M까지

//더 적은 동전 개수로 금액을 만들 수 있다면

if numCoin[i - coin] + 1 < numCoin[i]:

numCoin[i] ← numCoin[i - coin] + 1 // 더 적은 동전 개수로 업데이트

coinUsed[i] ← coin // i원을 만들 때 사용한 동전(coin) 기록

5. i ← M // (사용한 동전들 출력하기)

while i > 0:

print coinUsed[i] // i원을 만들 때 쓴 동전 출력

i ← i - coinUsed[i] // 해당 동전만큼 금액 줄이고 다음으로 이동

6. print numCoin[M] // 최소 동전 개수 출력하기

1. 텍스트, 폰트, 스크린샷, 친필이(가) 표시된 사진

   AI가 생성한 콘텐츠는 부정확할 수 있습니다.알고리즘의 수행시간 빅오 : O(dM)
2. 프로그램

**for (i = 0; i < d; i++) {**

**int coin = c[i];**

**for (j = coin; j <= M; j++) { // coin ~ M까지**

**if (numCoin[j - coin] + 1 < numCoin[j]) {**

**// 새로운 방법이 기존 방법보다 동전 개수를 덜 쓰는 경우**

**numCoin[j] = numCoin[j - coin] + 1;**

**coinUsed[j] = coin;**

**}**

**}**

**}**

1. 출력값 : 사용한 최소 동전 개수 3개 ( 사용한 동전은 5원 3원 3원)

텍스트, 폰트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

1. 설명

작은 문제부터 해결하며 그 결과를 저장하고 재활용해, 결국 큰 문제를 해결할 수 있도록 하는 bottom-up (dynamic programming) 특성을 활용했습니다.

M=11원을 만들기 위해, 주어진 1,3,5원으로, 1원을 만드는 것부터 시작해서 2원, …, M-1원을 만들어 보고 이때 사용한 동전 개수를 기록해, 만약 동전 개수를 덜 썼다면 덜 쓴 동전 개수로 값이 업데이트 되도록, 따라서 최종적으로 최소한으로 동전을 쓴 경우가 도출되도록 구현했습니다.

M: 만들고자 하는 목표 금액, c[] : 사용 가능한 동전들의 종류(1, 3, 5원), coin: coins[]배열에서 꺼내 현재 사용중인 동전 (e.g. 1이면 1, 3이면 3,5이면 5), numCoin[i] : i를 만들기 위해 사용한 최소 동전 개수(e.g. 2원 만들기 위해 동전 개수 2개 사용)를 의미합니다.

핵심 코드는 (3) 사진과 같습니다. 1,3,5 세 번 동안 루프를 도는데, 예를 들어 coin=1이면 1원짜리를 기준으로 1원부터 11(=M)원까지 만들 수 있는 경우를 모두 검사하고, coin=3이면 3원짜리를 기준으로 기존 만든 경우와 비교해 3원부터 11(=M)원까지 만들 수 있는 경우를 검사하고 (이때 동전 개수를 덜 사용했으면 덜 사용한 개수로 값을 업데이트), coin=5이면 5원짜리를 기준으로 기존에 만든 경우와 비교해 5원부터 11(=M)원까지 만들 수 있는 경우를 검사합니다 (이때도, 동전 개수를 덜 사용했으면 그 개수로 값을 업데이트). 그리고 이때 검사하는 코드 부분이 바로 coin~M까지 도는 for문(2번째 for문) 내에 if (numCoin[j - coin] + 1 < numCoin[j]) { numCoin[j] = numCoin[j - coin] + 1; coinUsed[j] = coin; }부분 입니다. 앞에서 j-coin원을 만든 전적이 있으니 거기서 동전 하나를 더 써 j원을 만든 경우(새로운 방법)와 지금까지 알고 있던 j원 만드는 방법(기존 방법)을 이용해 둘 중 어느 방법이 더 적은 동전 개수로 만들 수 있는지 비교하고, 동전 개수를 덜 쓴 경우로 동전 개수를 업데이트해 최종 결과가 나타나도록 한 부분입니다. coinUsed는 어떤 코인을 썼는지 알아보기 위해 만든 배열입니다.

**문제 2. 돌 문제**

1. 게임을 누가 먼저 시작하는지가 게임의 승패를 좌우하나요? (5점)

네. 누가 먼저 시작하는지가 게임의 승패를 좌우합니다. m개와 n개의 돌이 두 상자에 있는 상태에서 영희와 철수가 번갈아 돌을 가져가기 때문에 현재 상태(m,n)가 이기는 자리(Win)인지 지는 자리(L)인지에 따라 먼저 돌을 가져가는 영희가 이길 수도 있고, 후에 돌을 가져가는 철수가 이길 수 있기 때문입니다.

1. 텍스트, 도표, 라인, 그래프이(가) 표시된 사진

   AI가 생성한 콘텐츠는 부정확할 수 있습니다.Dynamic programming 을 사용하여 1~5 범위내의 m 과 n 에 대해서 이기는 전략 테이블을 이용 만들어 보시오. 예를 들어, m=5, n=3일 때 영희와 철수 중 누가 이길지 테이블을 보고 알 수 있어야 합니다. 힌트. 영희가 이길 때는 테이블에 W 를 적고 영희가 질때는 테이블에 L 를 적으시오. (20점)

현재 상태(i,j)의 승패가 이전 상태들의 결과를 참고해서 결정되었음을 보여주기 위해 화살표로 관계를 표현했습니다! (dynamic programming은 이전에 계산한 결과를 저장해두고 재사용하는 방식으로 다음 값을 빠르게 구할 수 있도록 하므로 이를 바탕으로 테이블과 화살표로 관계를 표현했습니다)

1. 유사 코드

Rocks(m, n): // m 과 n은 돌 개수

R[0][0] <- L // 돌이 아예 없으면 진 것 (lose)

for i = 1 to m: // 왼쪽(i) 상자에만 돌 있을 때 승패(W/L) 계산하기

if R[i-1][0] == W: // 상대(철수)가 이기면 영희는 진 것

R[i][0] <- L

else:

R[i][0] <- W

for j = 1 to n: //오른쪽(j) 상자에만 돌 있을 때 승패 계산

if R[0][j-1] == W: // 방식은 위와 동일

R[0][j] <- L

else:

R[0][j] <- W

// 두 상자에 모두 돌이 있는 경우

for i = 1 to m:

for j = 1 to n:

// R[i-1][j]: 왼쪽에서 돌 하나 뺀 것

// R[i][j-1]: 오른쪽에서 하나 돌 뺀 것

// R[i-1][j-1: 양쪽에서 하나씩 뺀 것 이 셋 전부 철수가 이기면 영희는 짐

if R[i-1][j-1] == W and R[i-1][j] == W and R[i][j-1] == W:

R[i][j] <- L

else: // 반대로 하나라도 이긴다면 영희 WIN

R[i][j] <- W

1. 프로그램 m=10, n=7일 때 승자 출력 : 영희 Win!

텍스트, 스크린샷, 흑백, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

영희가 먼저 돌을 꺼냈을 때 각 상태 (m, n)에 대한 영희/철수 승패를 계산하는 table을 만들었습니다. 영희와 철수가 돌을 번갈아 돌을 꺼내므로 R[i][j]는 이전 상태인 ( i-1, j ), ( i, j-1 ), ( i-1, j-1 ) 값을 재활용해 dynamic programming이 적용되도록 구현했습니다. 여기서, R[i][j]는 해당 상태에서 차례인 사람이 이기는지 여부를 의미합니다.