CS202 Topic #3 Report

전산학부
20170038 권기훈

[tomato]

(1, 1, 1,)부터 (H,M,N)까지 H*M*N번 반복문을 이용하여 그 지점의 가격을 먼저 계산하였고(price vector), 같은 크기의 벡터를 만들어서 (1, 1, 1,)부터 (H,M,N)까지의 누적합을 구해줬습니다.

누적합을 구하는 방식은 기본적으로 1차원, 2차원에서 했던 방식을 확장하였습니다. 포함-배제의 원리를 이용하였으며, 바로 그 값을 구할 수는 없기 때문에, x방향, y방향, z방향의 1차원의 누적합 값들을 먼저 계산하고, xy평면, yz평면, zx평면의 누적합을 계산한 뒤에 다시 N*M*H번 반복문을 이용하여 모든 좌표의 누적합을 계산하였습니다. 그 코드는 아래와 같습니다.

```
priceSum[i][j][k] = price[i][j][k] + priceSum[i - 1][j][k] + priceSum[i][j - 1][k] + priceSum[i][j][k - 1] \
- priceSum[i - 1][j - 1][k] - priceSum[i - 1][j][k - 1] - priceSum[i][j - 1][k - 1] +priceSum[i - 1][j -1][k - 1];
```

그 뒤에 구간합을 구해야 할 각 좌표 값을 받은 후에는 역시 다시 포함-배제의 원리를 이용하여 실제 필요한 구간을 계산하였습니다. 그 코드는 아래와 같습니다.

```
soldPrice = priceSum[x2][y2][z2] - priceSum[x1 - 1][y2][z2] - priceSum[x2][y1 - 1][z2] - priceSum[x2][y2][z1 - 1] + \
priceSum[x1 - 1][y1 - 1][z2] + priceSum[x1 - 1][y2][z1 - 1] + priceSum[x2][y1 - 1][z1 - 1] - priceSum[x1 - 1][y1 - 1][z1 - 1];
```

알고리즘의 시간복잡도는 O(N*M*H)으로 기대할 수 있습니다.

[bigdata]

2번째 조건까지 수행하는 코드입니다. N, M <= 250이고 클럭 속도가 모두 1인 상황이므로 이를 knapsack 문제로 대응시켜서 생각할 수 있습니다.

문제의 상황처럼 컴퓨터를 빌리고 작업을 수행하는 것을 각각 하는 것보다는
일단 모든 컴퓨터를 다 빌리고 빌린 컴퓨터에 해당하는 성능의 작업을 추가한 뒤 선택적으로 수행한다면 결과론적으로는 문제의 상황과 같아집니다.

따라서 이를 knapsack 문제에 대응하면
작업의 수 = 가방의 넣을 물건의 수
빌릴 수 있는 컴퓨터의 총 코어의 수 = 가방의 크기
최종적으로 profit[N+M-1][CoreNum] 의 값을 계산해주면 됩니다.

이 알고리즘의 시간복잡도는 O(50*N(N+M)) (50이 상당히 유의미한 숫자가 된다고 생각합니다.) 정도로 기대할 수 있습니다.

클럭 속도가 다를 경우에는 작업과 컴퓨터 성능을 다 비교해주어야 하나, 주어진 시간 조건 안에 이를 만족시키는 점화식을 찾지 못하였습니다..