Question 2

Solution. **Step 0**: Recursive substructure. In order to find the optimum path, we must find the maximum potential amount of gold we can get from drilling to each block. In order to do this for a given block (i, j), we need to look at the potential amount of gold we get for the 3 blocks above it from which we can then drill down to (i, j). The base case is the surface layer, where the potential amount of gold is just the amount of gold in the block if the hardness of the block is at most d, and 0 otherwise.
Step 1: Array definition.

For ease of notation, we define two $m \times n$ arrays A and D, so that $A[i, j]$ is the maximum amount of gold obtainable at location (i, j) in the mine, while $D[i, j]$ represents the amount of hardness of the drill remaining at position (i, j) by following the same path. If the remaining drill hardness is not enough to harness position (i, j) of the mine, we set $A[i, j] = D[i, j] = -\infty$. Notice that at the first row ($i = 1$), the remaining drill hardness is just the original drill hardness d.

Step 2: Array recurrence.

$$D[1, j] = \begin{cases} d - H[1, j], & \text{if } H[1, j] \leq d \\ -\infty, & \text{otherwise.} \end{cases}$$

$$A[1, j] = \begin{cases} G[1, j], & \text{if } H[i, j] \leq d \\ -\infty, & \text{otherwise.} \end{cases}$$

For $i > 1$ and $1 \leq j \leq n$, define X as the set of *indices* such that $x \in X$ means $j - 1 \leq x \leq j + 1$ (it's a valid previous "x" co-ordinate) and $H[i, j] \leq D[i - 1, x]$ (enough drill-hardness remained at that slot to mine the current square). If $X = \varnothing$, we define $D[i, j] = A[i, j] = -\infty$. Otherwise, set y as,

$$\operatorname*{argmax}_{x \in X} A[i - 1, x],$$

i.e. y is the "x" co-ordinate of the previous valid square at which we had the most gold. Then we define $D[i, j]$ as,

$$D[i, j] = D[i - 1, y] - H[i, j].$$

Similarly, we define $A[i, j]$ as,

$$A[i, j] = A[i - 1, y] + G[i, j].$$

Step 3: Iterative algorithm. This populates A, D (and S so we can reconstruct the optimum solution).

```
OPTIMUM-GOLD(G, H, d):
 1  let A[1..m, 1..n] and D[1..m, 1..n] be new m × n matrices. // the arrays defined above.
 2  let S[1..m, 1..n] be a new m × n matrix // stores how we got to (i, j)
 3  for i = 1 to m:
 4      for j = 1 to n:
 5          A[i, j] = −∞
 6          D[i, j] = −∞
 7          if i == 1:
 8              if H[i, j] ≤ d: // sufficient drill hardness
 9                  A[i, j] = G[i, j]
10                  D[i, j] = d − H[i, j]
11                  S[i, j] = j
12          else :
13              best = −∞
14              for x = j − 1 to j + 1: // possible parent slots
15                  if 1 ≤ x ≤ n and A[i − 1, x] ≥ best and H[i, j] ≤ D[i − 1, x]: // we can mine (i, j)
16                      best = A[i − 1, x]
17                      A[i, j] = A[i − 1, x] + G[i, j]
18                      D[i, j] = D[i − 1, x] − H[i, j]
19                      S[i, j] = x
20  return (A, S)
```

Step 4: Reconstructing the optimum solution.

To reconstruct an optimum solution, we simply need to first find the slot of the mine that yields the most total gold (while using at most the original drill hardness d). The maximum amount of gold is easily found, since we simply have to scan A for its largest value, and this also gives us the corresponding position (i, j) – the last element of the optimum path. Once we have found this position, we simply use S to go backwards (up the mine) and find the previous element of the path, until we reach the top of the mine, and this gives us the desired optimum path. The procedure RECONSTRUCT-OPTIMUM-PATH(A, S) implements precisely this idea.

```
RECONSTRUCT-OPTIMUM-PATH(A, S):
 1  k = 0 // length of the optimum path
 2  maxGold = 0
 3  j' = 0
 4  for i = 1 to m:
 5      for j = 1 to n:
 6          if A[i, j] > maxGold:
 7              maxGold = A[i, j]
 8              k = i
 9              j' = j
10  let J be a new k element array // the entire k-element path
11  for i = k to 1: // populating J backwards
12      J[i] = j'
13      j' = S[i, j'] // the previous element of the path
14  return J
```

♦