CSE 417
Dynamic Programming (pt 1)
Definition, History, & Simple Examples

UNIVERSITY *of* WASHINGTON

Reminders

- **HW4 is posted: due in one week**
  - start early!

Outline for Today

> Motivation & Definition
> Robot Example
> History
> Robot Example 2
> Extensions

Dynamic Programming: Motivation

Dynamic programming is...

- most useful algorithm design technique in practice (IMO)
- more robust to problem changes than techniques discussed so far
- usually easiest to analyze for run-time performance
- often easier to implement
  - some could be implemented in Excel
- ubiquitous in CS
  - more so than greedy or divide & conquer
  - applications are large in number and *importance*

Dynamic Programming: Applications

- compilers
  - optimal code generation
- machine learning
  - speech recognition
  - parsing natural language
- databases
  - query optimization
- graphics
  - optimal polygon triangulation
- networking
  - routing
- practical applications:
  - spell checking
  - file comparison
  - document layout
  - pattern matching
- many, many more...

Dynamic Programming vs Divide & Conquer

	Divide & Conquer	**Dynamic Programming**
Sub-problems	split data in parts	**any subsets of input** – we chose which ones – (not all subsets)
Enumeration Strategy	recursion	

Both apply "optimal substructure":
using solution on sub-problems to solve whole problem

Dynamic Programming vs Divide & Conquer

	Divide & Conquer	**Dynamic Programming**
Sub-problems	split data in parts	**any subsets of input** – we chose which ones
Enumeration Strategy	recursion	**solve them all** – often record in a table

Dynamic Programming: Definition

Dynamic programming approach…

> describe solution in terms of solution to sub-problems
  – like D&C but consider more general subsets of data
> solve *every* sub-problem we need
  – once only!

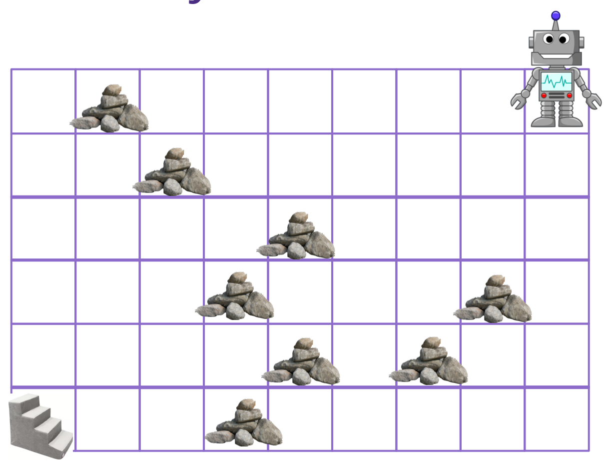
> Approach is efficient if there aren't too many sub-problems
  – often not trying to get from $O(n^2)$ to $O(n \log n)$ as in D&C… trying to get from impossible to possible
  – number of sub-problems could be large but want $n^{O(1)}$

Outline for Today

> Motivation & Definition
> Robot Example
> History
> Robot Example 2
> Extensions

Example: Broken Robot Path

> **Problem**: Given an n x m grid where some squares contain rocks, find a path for the robot to get from (n,m) to the exit at (1,1), where the robot only has to move *down or left.*



Example: Broken Robot Path

- **Problem**: Given an n x m grid where some squares contain rocks, find a path for the robot to get from (n,m) to the exit at (1,1), where the robot only has to move *down or left.*

- Problem is too easy so far…
  - Q: how do you solve it efficiently?

Example: Broken Robot Path

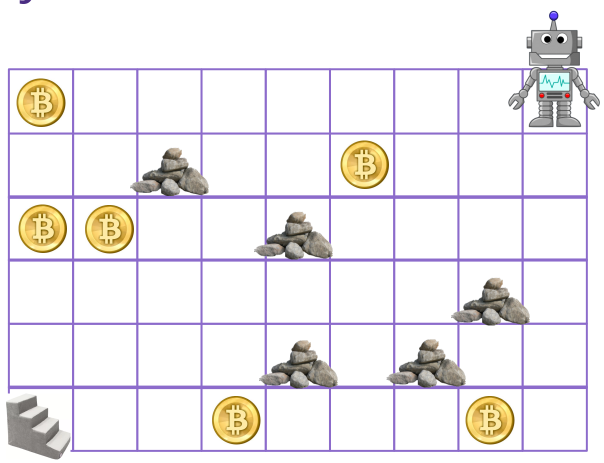
- **Problem**: Given an n x m grid where some squares contain rocks, find a path for the robot to get from (n,m) to the exit at (1,1), where the robot only has to move *down or left.*

- Problem is too easy so far…
  - a shortest path problem

- Even easier interview question: no rocks
  - problem was to count the number of paths
  - can be solved by dynamic programming, but…

Example: Bitcoin Mining Broken Robot

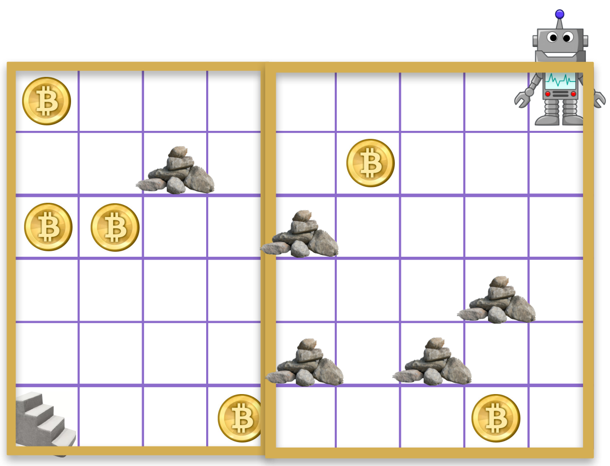
> **Problem**: Given an n x m grid where squares have rocks or bitcoins, find a path for the robot to get from (n,m) to the exit at (1,1), where the robot only has to move down or left, *maximizing the coins found*!



now an optimization problem

Example: Bitcoin Mining Broken Robot

- **Q**: What are the sub-problems?
  - have to choose these for D&C or DP



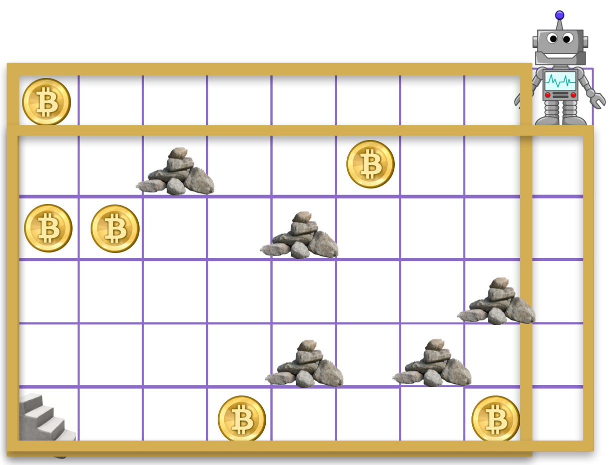
Divide & Conquer would split the input into pieces...

Doesn't seem useful...

- e.g., right box finds opt **path ending at (5,1)**
- solution does not use that

Example: Bitcoin Mining Broken Robot

> **Q**: What are the sub-problems?
  – DP allows more general choices



both find paths to (1,1)
but with different starts

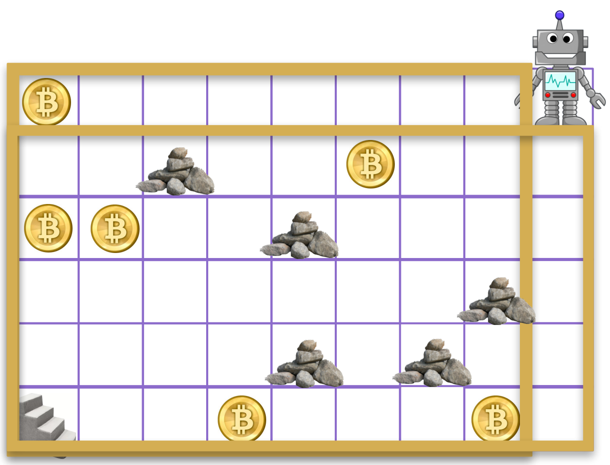
These are useful…

> every path from (n,m) steps through (n-1,m) or (n,m-1)

> opt must do one or other

Example: Bitcoin Mining Broken Robot

- > **Q**: What are the sub-problems?
  - – DP allows more general choices



Optimal path from (n,m) either goes through (n-1,m) or (n,m-1).

max coin collected from (n,m)
= max(max coin from (n-1,m),
max coin from (n,m-1))
+ 1 if coin at (n,m)
(or = 0 if rocks at (n,m))

Example: Bitcoin Mining Broken Robot

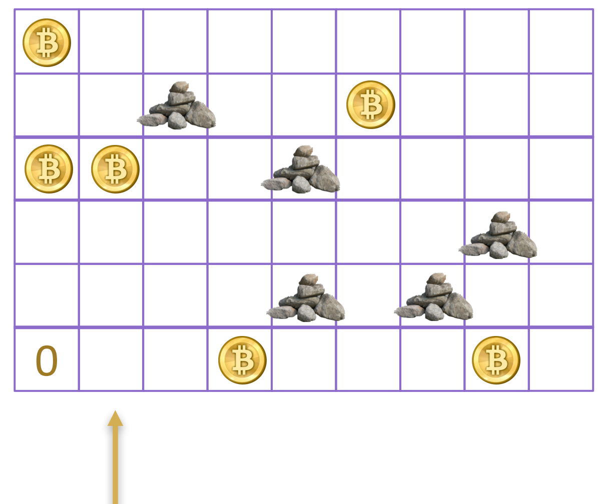
- **Q**: What are the sub-problems?
- **A**: Choice of starting point (top-right of rectangle)
  - all rectangles with bottom-left at (1,1)
  - same formula applies to find opt at any (i,j)

- **Q**: How many sub-problems are there?
- **A**: nm
  - running time will be nm x time per sub-problem

Example: Bitcoin Mining Broken Robot

> **Q**: What are the sub-problems?
> **A**: Choice of starting point

> Algorithm:
  - allocate a table A with n x m entries
  - record the optimal solution in each spot
  - fill the table in from bottom-left to top-right



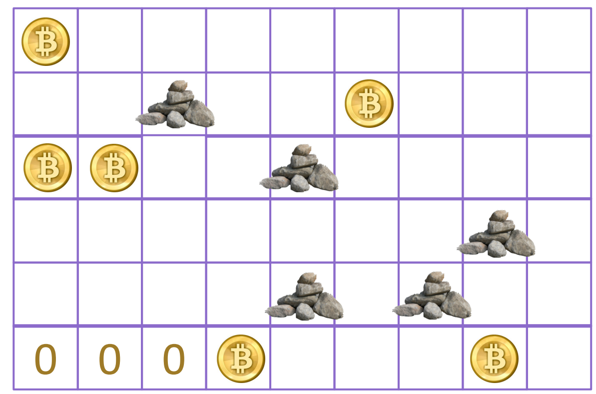
no solution below, so
opt = left + 1 if coin here

Example: Bitcoin Mining Broken Robot

> **Q**: What are the sub-problems?

> **A**: Choice of starting point

> Algorithm:
  - allocate a table A with n x m entries
  - record the optimal solution in each spot
  - fill the table in from bottom-left to top-right

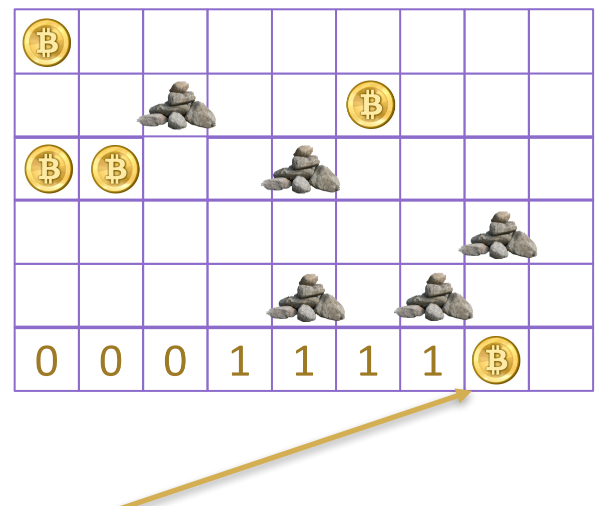


no solution below, so
opt = left + 1 if coin here

Example: Bitcoin Mining Broken Robot

- **Q**: What are the sub-problems?
- **A**: Choice of starting point

- Algorithm:
  - allocate a table A with n x m entries
  - record the optimal solution in each spot
  - fill the table in from bottom-left to top-right



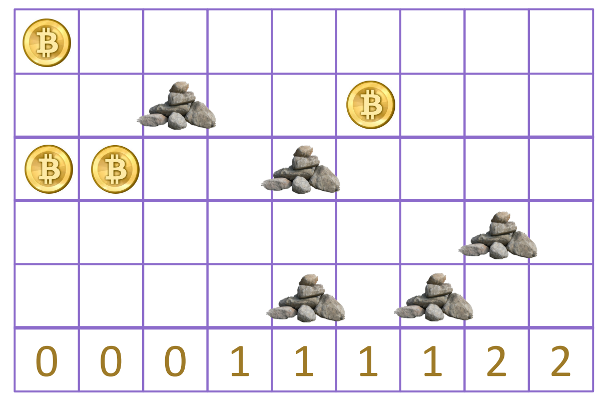
no solution below, so
opt = left + 1 if coin here

Example: Bitcoin Mining Broken Robot

> **Q**: What are the sub-problems?

> **A**: Choice of starting point

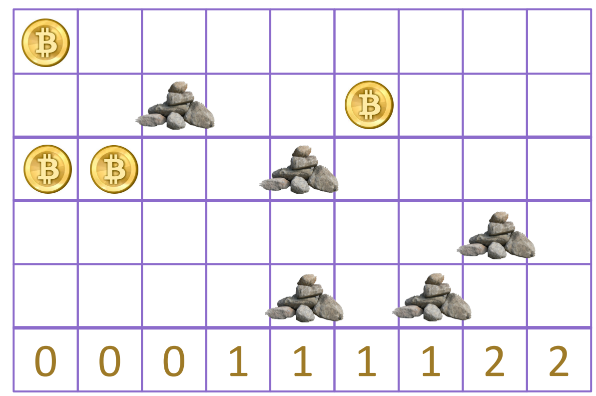
> Algorithm:
  - allocate a table A with n x m entries
  - record the optimal solution in each spot
  - fill the table in from bottom-left to top-right



Example: Bitcoin Mining Broken Robot

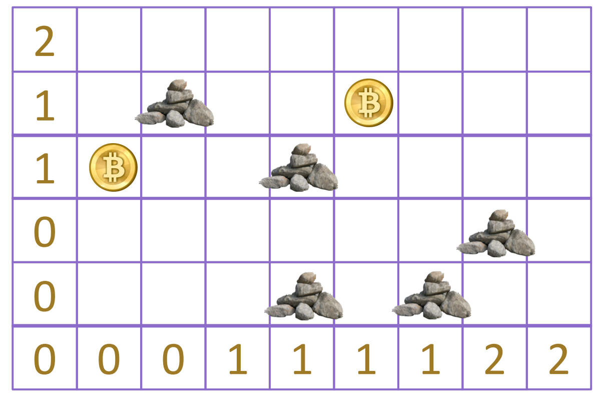
- **Q**: What are the sub-problems?
- **A**: Choice of starting point

no solution to left, so
opt = below + 1 if coin here



Example: Bitcoin Mining Broken Robot

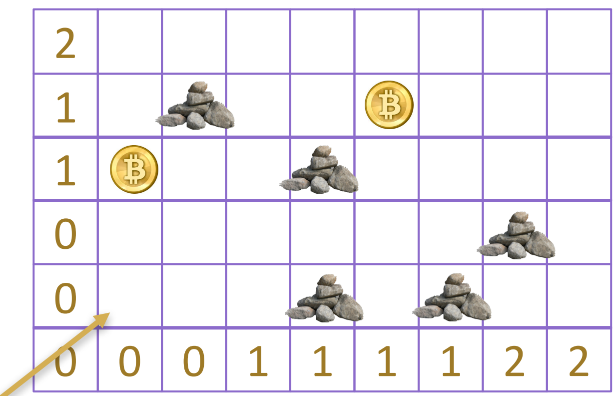
> **Q**: What are the sub-problems?
> **A**: Choice of starting point



> Algorithm:
  – allocate a table A with n x m entries
  – record the optimal solution in each spot
  – fill the table in from bottom-left to top-right
    > usually fill in the special (edge) cases first.
    > many possible orders (row at a time, col at a time, diagonals, etc.)
    > only rule is bottom & left must be filled in first

Example: Bitcoin Mining Broken Robot

> **Q**: What are the sub-problems?
> **A**: Choice of starting point

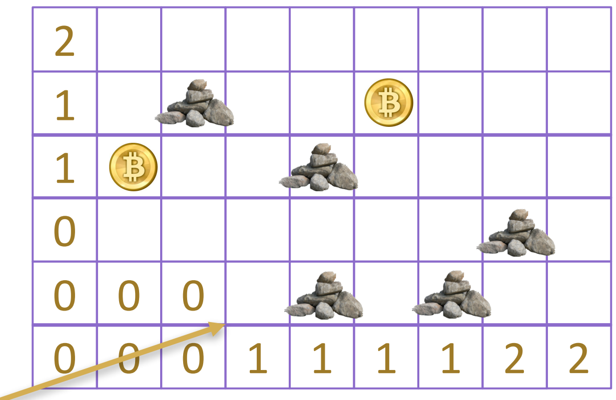


opt here = max(opt left, opt below)
+ 1 if coin here

Example: Bitcoin Mining Broken Robot

> **Q**: What are the sub-problems?

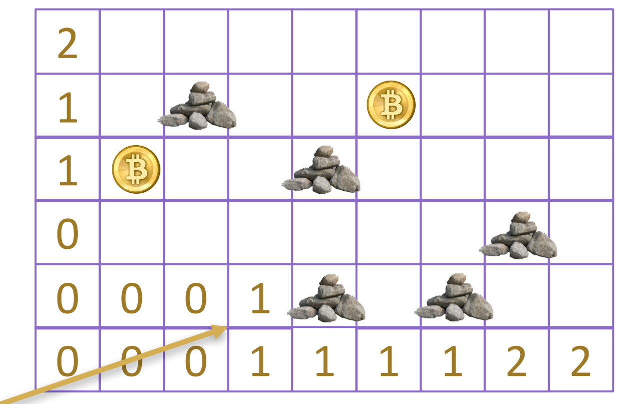
> **A**: Choice of starting point



opt here = max(opt left, opt below)
+ 1 if coin here

Example: Bitcoin Mining Broken Robot

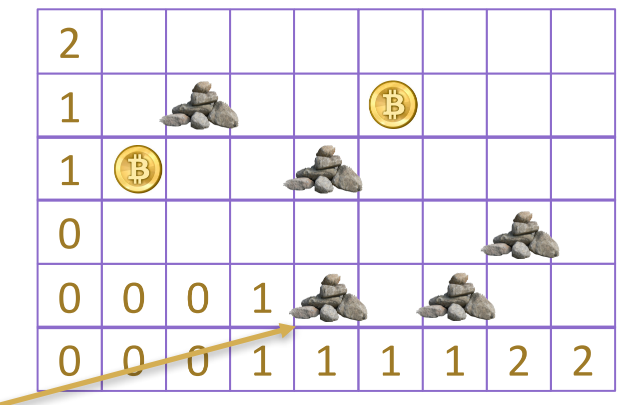
- **Q**: What are the sub-problems?
- **A**: Choice of starting point



opt here = max(opt left, opt below)
+ 1 if coin here

Example: Bitcoin Mining Broken Robot

- **Q**: What are the sub-problems?
- **A**: Choice of starting point

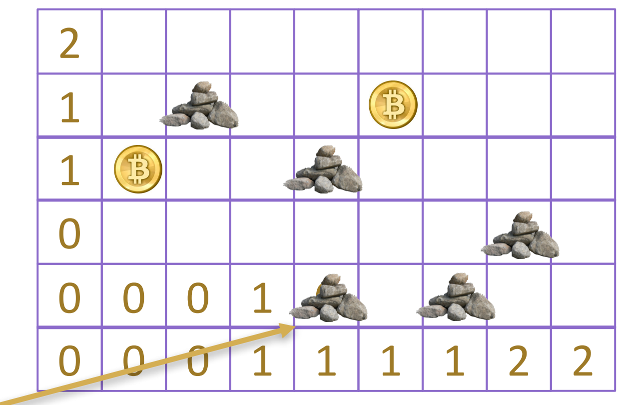


opt here = max(opt left, opt below)
+ 1 if coin here
opt here = 0 if rocks (or maybe -1?)

Example: Bitcoin Mining Broken Robot

> **Q**: What are the sub-problems?

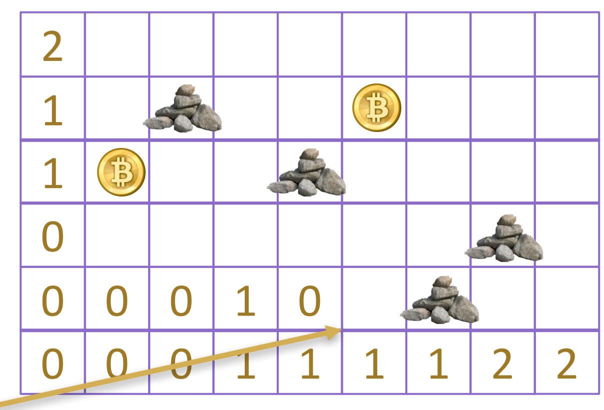
> **A**: Choice of starting point



opt here = max(opt left, opt below)
+ 1 if coin here
opt here = 0 if rocks (or maybe -1?)

Example: Bitcoin Mining Broken Robot

> **Q**: What are the sub-problems?
> **A**: Choice of starting point

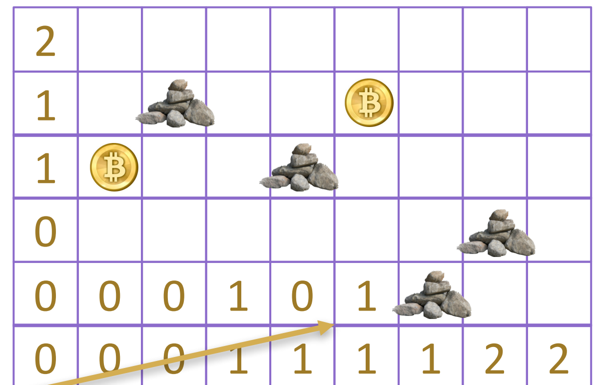


opt here = max(opt left, opt below)
+ 1 if coin here
opt here = 0 if rocks   (or maybe -1?)

Example: Bitcoin Mining Broken Robot

> **Q**: What are the sub-problems?

> **A**: Choice of starting point

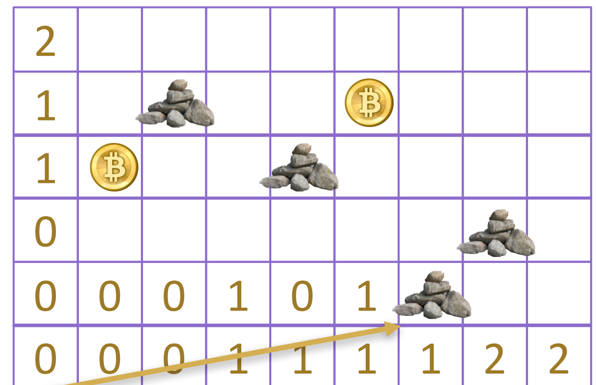


opt here = max(opt left, opt below)
+ 1 if coin here
opt here = 0 if rocks (or maybe -1?)

Example: Bitcoin Mining Broken Robot

> **Q**: What are the sub-problems?

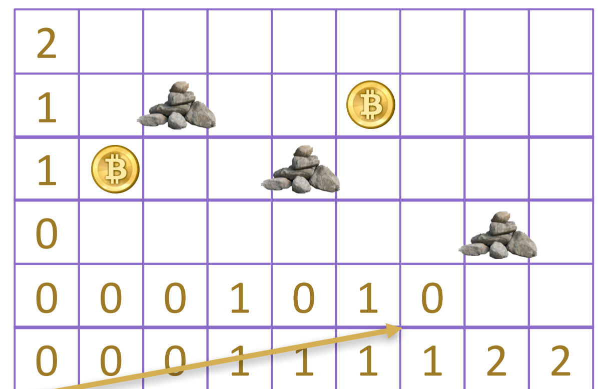
> **A**: Choice of starting point



opt here = max(opt left, opt below)
+ 1 if coin here
opt here = 0 if rocks (or maybe -1?)

Example: Bitcoin Mining Broken Robot

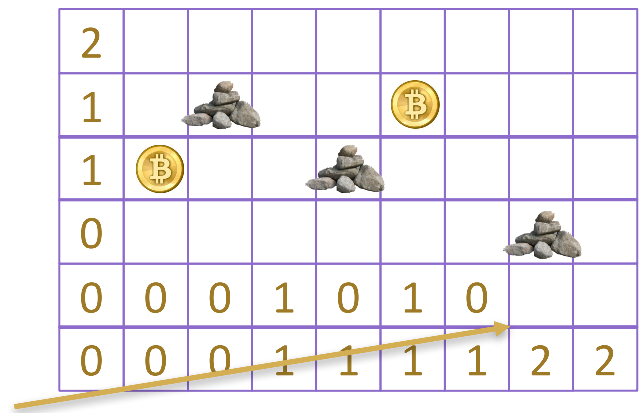
> **Q**: What are the sub-problems?
> **A**: Choice of starting point



opt here = max(opt left, opt below)
+ 1 if coin here
opt here = 0 if rocks (or maybe -1?)

Example: Bitcoin Mining Broken Robot

> **Q**: What are the sub-problems?
> **A**: Choice of starting point



opt here = max(opt left, opt below)
 + 1 if coin here
opt here = 0 if rocks   (or maybe -1?)

Example: Bitcoin Mining Broken Robot

- > **Q**: What are the sub-problems?
- > **A**: Choice of starting point

2								
1								
1								
0								
0	0	0	1	0	1	0	2	2
0	0	0	1	1	1	1	2	2

opt here = max(opt left, opt below)
 + 1 if coin here
opt here = 0 if rocks   (or maybe -1?)

Example: Bitcoin Mining Broken Robot

- **Q**: What are the sub-problems?
- **A**: Choice of starting point

2	2	2	2	2	3	3	3	3
1	2	0	2	2	3	3	3	3
1	2	2	2	0	1	1	1	2
0	0	0	1	1	1	1	0	2
0	0	0	1	0	1	0	2	2
0	0	0	1	1	1	1	2	2

opt here = max(opt left, opt below)
+ 1 if coin here
opt here = 0 if rocks (or maybe -1?)

Example: Bitcoin Mining Broken Robot

- **Q**: What are the sub-problems?
- **A**: Choice of starting point

- Running time is O(1) per entry

  opt here = max(opt left, opt below)
  + 1 if coin here
  opt here = 0 if rocks

- Total running time is O(nm)

2	2	2	2	2	3	3	3	3
1	2	0	2	2	3	3	3	3
1	2	2	2	0	1	1	1	2
0	0	0	1	1	1	1	0	2
0	0	0	1	0	1	0	2	2
0	0	0	1	1	1	1	2	2

Example: Bitcoin Mining Broken Robot

```
int[][] A = new int[n+1][m+1];

A[1][1] = 0;

for (int i = 2; i <= n; i++) {
  if (B[i][1] == COIN)
    A[i][1] = A[i-1][1] + 1;
  else if (B[i][1] == ROCK)
    A[i][1] = -Infinity;
  else
    A[i][1] = A[i-1][1];
}

// ... fill in A[1][j] similarly ...
```

```
for (int i = 2; i <= n; i++) {
  for (int j = 2; j <= m; j++) {
    if (B[i][j] == COIN)
      A[i][j] = 1 +
          max(A[i-1][j], A[i][j-1]);
    else if (B[i][j] == ROCK)
      A[i][j] = -Infinity;
    else
      A[i][j] = max(A[i-1][j], A[i][j-1]);
  }
}

return A[n][m];
```

Outline for Today

> Motivation & Definition
> Robot Example
> History
> Robot Example 2
> Extensions

Dynamic Programming: Etimology

> *programming* : program :: scheduling : schedule
  - think of a program for a concert
  - choice of what to play and when to play it (not just a schedule)
  - same use as “linear programming”, “convex programming”, etc.

> *dynamic* means relating to time
  - inventor (Bellman) was looking at problems where index was time
    > e.g., our price data in HW4
  - BUT **time plays no role** in modern user of the word

Dynamic Programming: History

> Technique invented by Richard Bellman in the 1950s
  – we will see the algorithm when we discuss network flows…

> At the time, Secretary of Defense did not like math research, so Bellman chose a name that did not sound like math
  – ”it is impossible to use the word ‘dynamic’ in a pejorative sense”
  – “[dynamic programming] was a name not even a Congressman could object to
  – (both quotes from Bellman’s autobiography)

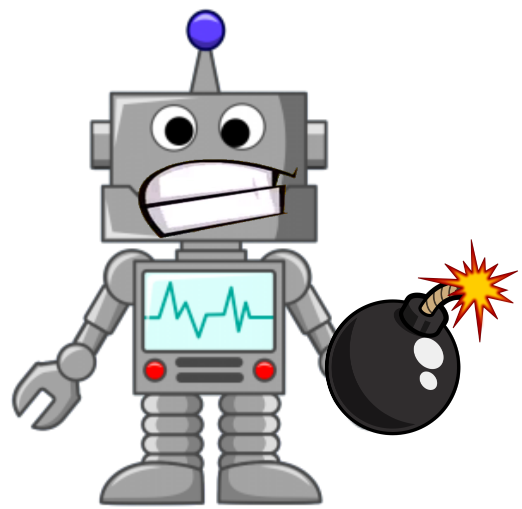
Outline for Today

- > Motivation & Definition
- > Robot Example
- > History
- > Robot Example 2
- > Extensions

Example: Bitcoin Mining Bomber Robot

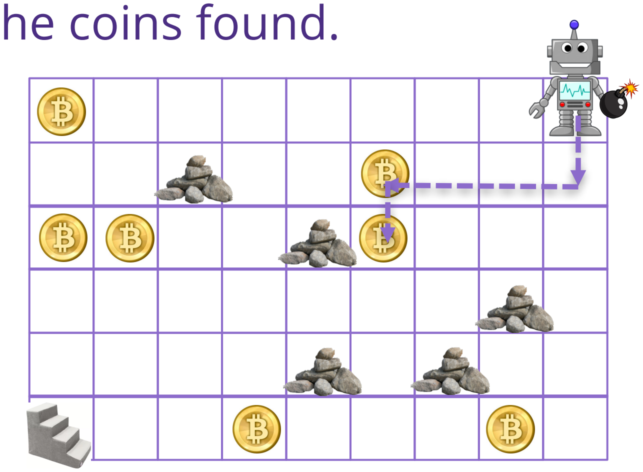
> Robot is tired of these rocks in his way!
  – he wants bitcoin!!!

> Robot buys a bomb he can use to blast rocks
  – only has one bomb
  – has to choose carefully where to use it…



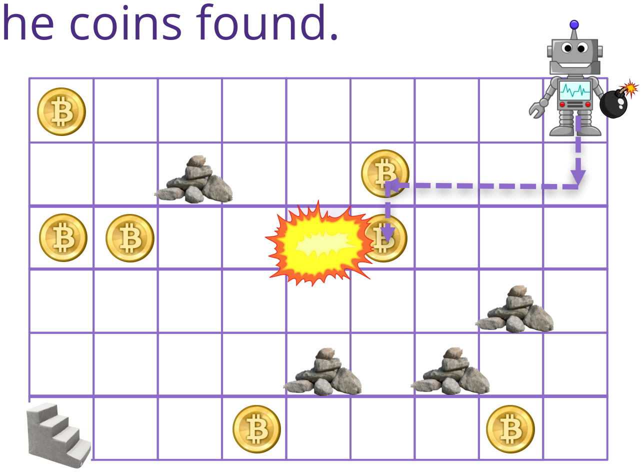
Example: Bitcoin Mining Bomber Robot

> **Problem**: Given an n x m grid where squares have rocks or <u>bitcoins</u>, find a path for the robot to get from (n,m) to the exit at (1,1), where the robot only has to move down or left or blast down or blast left (*one time only*), maximizing the coins found.



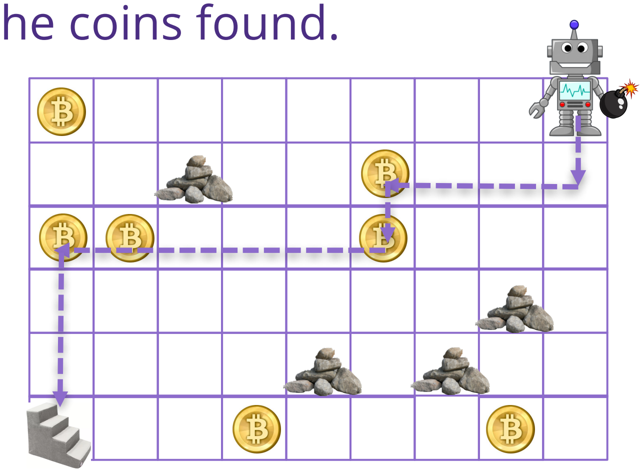
Example: Bitcoin Mining Bomber Robot

> **Problem**: Given an n x m grid where squares have rocks or bitcoins, find a path for the robot to get from (n,m) to the exit at (1,1), where the robot only has to move down or left or blast down or blast left (*one time only*), maximizing the coins found.

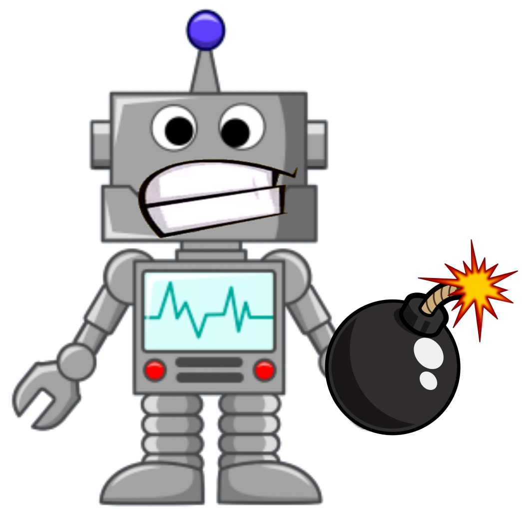


Example: Bitcoin Mining Bomber Robot

- **Problem**: Given an n x m grid where squares have rocks or bitcoins, find a path for the robot to get from (n,m) to the exit at (1,1), where the robot only has to move down or left or blast down or blast left (*one time only*), maximizing the coins found.



Example: Bitcoin Mining Bomber Robot



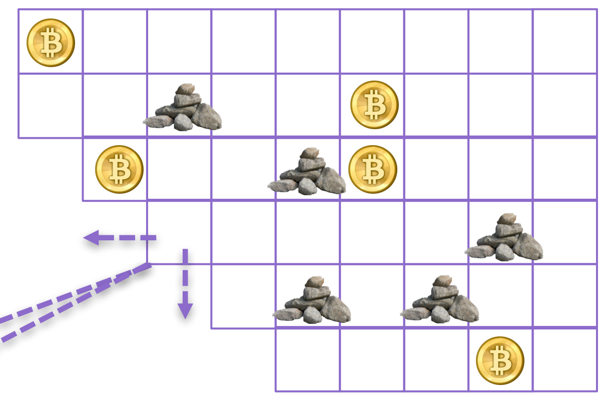
- **Q**: What are the sub-problems?
- **A**: Choice of starting point &
  whether he still has bomb

- Can compute this using two tables:
  - one for opt solution with no bomb (saw before)
  - one for opt solution with one bomb left
    - opt at (i,j) with bomb = max(
      opt at (i-1,j) with bomb, opt at (i,j-1) with bomb,
      opt at (i-1,j) using bomb, opt at (i,j-1) using bomb)
    - opt at (i,j) using bomb =
      formula for opt at (i,j) without bomb ignoring rocks there

Example: Bitcoin Mining Bomber Robot

> **Q**: What are the sub-problems?

> **A**: Choice of starting point & whether he still has bomb



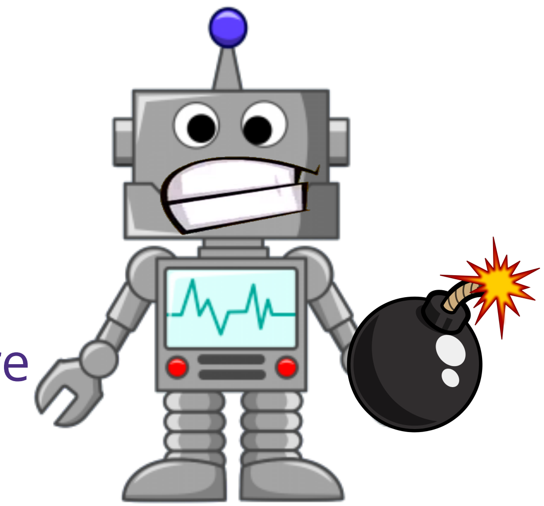
can move within one table or jump to the other table…

2	2	2	2	2	3	3	3	3
1	2	0	2	2	3	3	3	3
1	2	2	2	0	2	2	2	2
0	0	0	1	1	1	1	0	2
0	0	0	1	0	1	0	2	2
0	0	0	1	1	1	1	2	2

BUT re-compute from down & left ignoring any rock here
(still O(1) time to compute)

Example: Bitcoin Mining Bomber Robot

- As in the example, it is often easy to accommodate small changes to problem
  - more so than greedy or divide & conquer
- Only doubles the number of sub-problems here
  - you will see similar situations in future HWs



Outline for Today

> Motivation & Definition
> Robot Example
> History
> Robot Example 2
> Extensions

Dyn Programming: Counting Solutions

- It is also possible to count solutions.
- Instead of storing just opt achievable in A[i][j],
  store the opt achievable and the number of solutions achieving it
  - if left is better, then #opt solutions is #opt solutions from left
  - if down is better, then #opt solutions is #opt solutions from down
  - if both are equal, then #opt solutions is…
    (#opt solutions from left) + (#opt solutions from down)
- Similar approach works for most DP algorithms

Dyn Programming: Finding Solutions

> Previous algorithm computed <u>value</u> of optimal solution
> BUT what if we want the solution that is optimal?

> Can get that from the table as well
  - walk from the end back to the beginning
  - follow along choices that achieve the max score

Dyn Programming: Finding Solutions

> Get optimal solution from table of optimal values…

2	2	2	2	2	3	3	3	3
1	2	0	2	2	3	3	3	3
1	2	2	2	0	2	2	2	2
0	0	0	1	1	1	1	0	2
0	0	0	1	0	1	0	2	2
0	0	0	1	1	1	1	2	2

Table says 3 is possible moving either down or right

Let's go down…

Dyn Programming: Finding Solutions

> Get optimal solution from table of optimal values...

2	2	2	2	2	3	3	3	3
1	2	0	2	2	3	3	3	3
1	2	2	2	0	2	2	2	2
0	0	0	1	1	1	1	0	2
0	0	0	1	0	1	0	2	2
0	0	0	1	1	1	1	2	2

Table says we must go left to get 3...

Dyn Programming: Finding Solutions

> Get optimal solution from table of optimal values…

2	2	2	2	2	3	3	3	3
1	2	0	2	2	3	3	3	3
1	2	2	2	0	2	2	2	2
0	0	0	1	1	1	1	0	2
0	0	0	1	0	1	0	2	2
0	0	0	1	1	1	1	2	2

Now neither solution achieves 3? What?

There's a coin at this spot!
Pick it up and look for 2 more, not 3 more.

Dyn Programming: Finding Solutions

> Get optimal solution from table of optimal values...

2	2	2	2	2	3	3	3	3
1	2	0	2	2	3	3	3	3
1	2	2	2	0	2	2	2	2
0	0	0	1	1	1	1	0	2
0	0	0	1	0	1	0	2	2
0	0	0	1	1	1	1	2	2

Dyn Programming: Finding Solutions

> Previous algorithm computed value of optimal solution
> BUT what if we want the solution that is optimal?

> Can get that from the table as well
> – walk from the end back to the beginning
> – follow along choices that achieve the max score

> Alternatively, keep track of how you got the value too
> – e.g., in robot, record if max was from down or left
> – requires extra space in the table