Dynamic Programming

Dynamic Programming

- A misleading name
  - Not a way of programming
  - A strategy for solving certain algorithmic problems

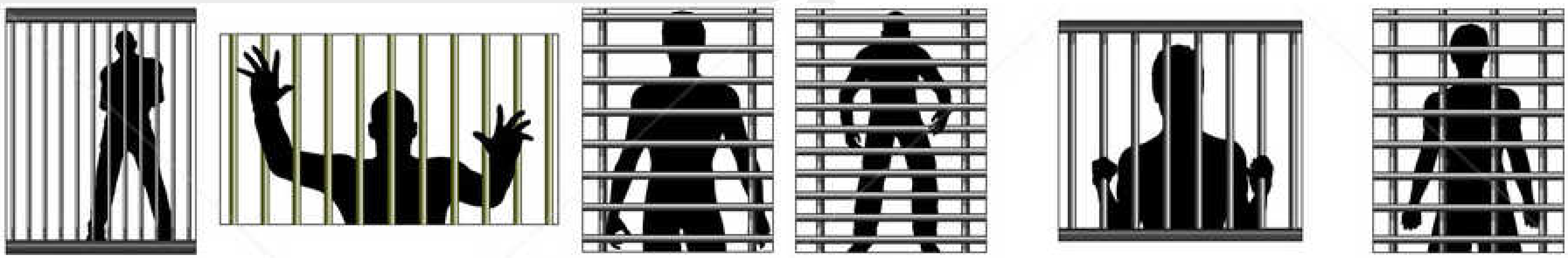
Approaches

- Top-down
    - Solve subproblems and remember those solved
- Bottom-up
    - From the smallest subproblems build solutions to larger problems

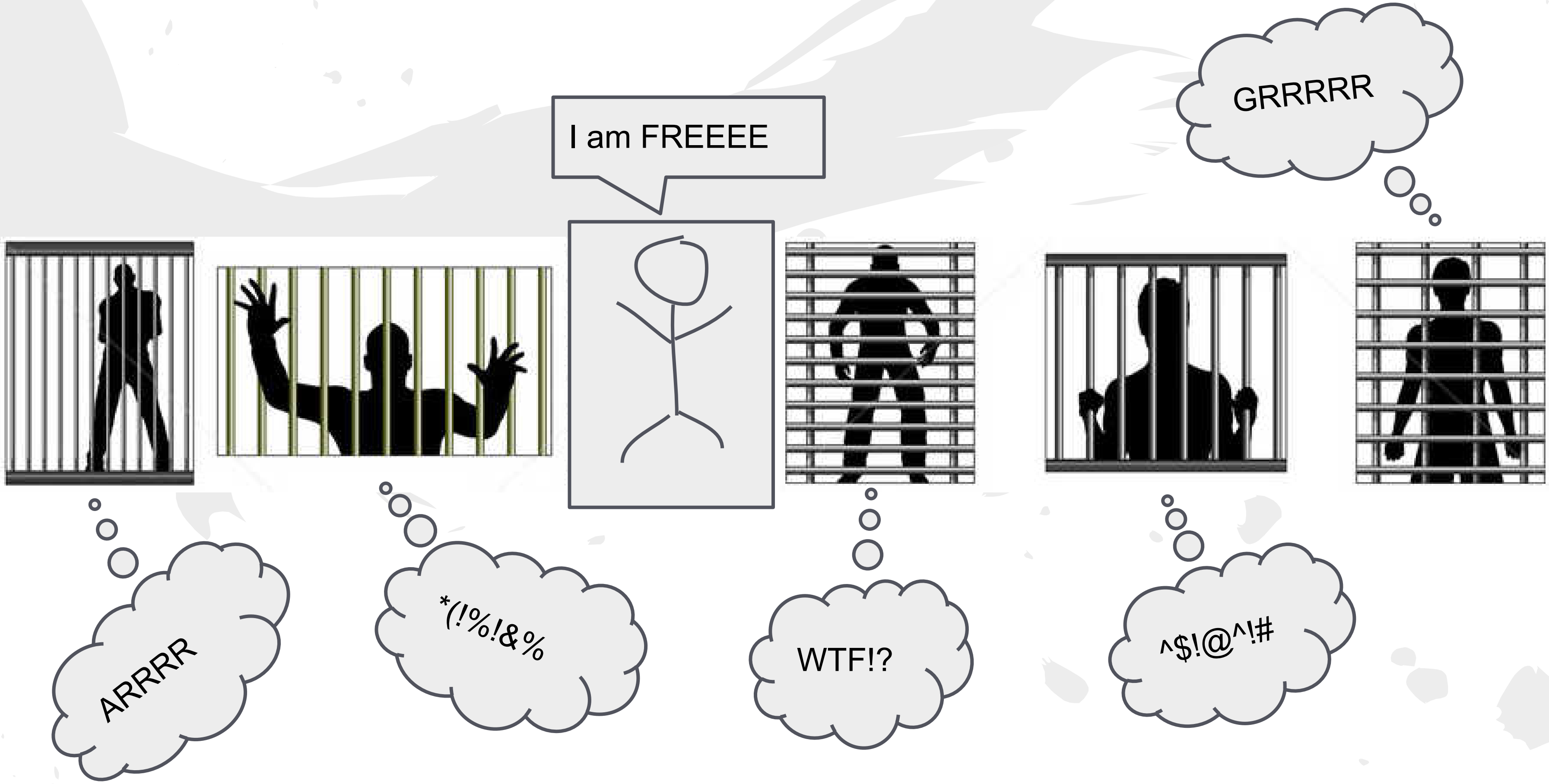
Problem Properties

- Optimal Substructure
    - Seek the optimal subproblem
        - i.e maximise a value with minimal cost
- Overlaping subproblems
    - Problem can be divided to subproblems
        - i.e to calculate Fib(5) we must calculate Fib(4)

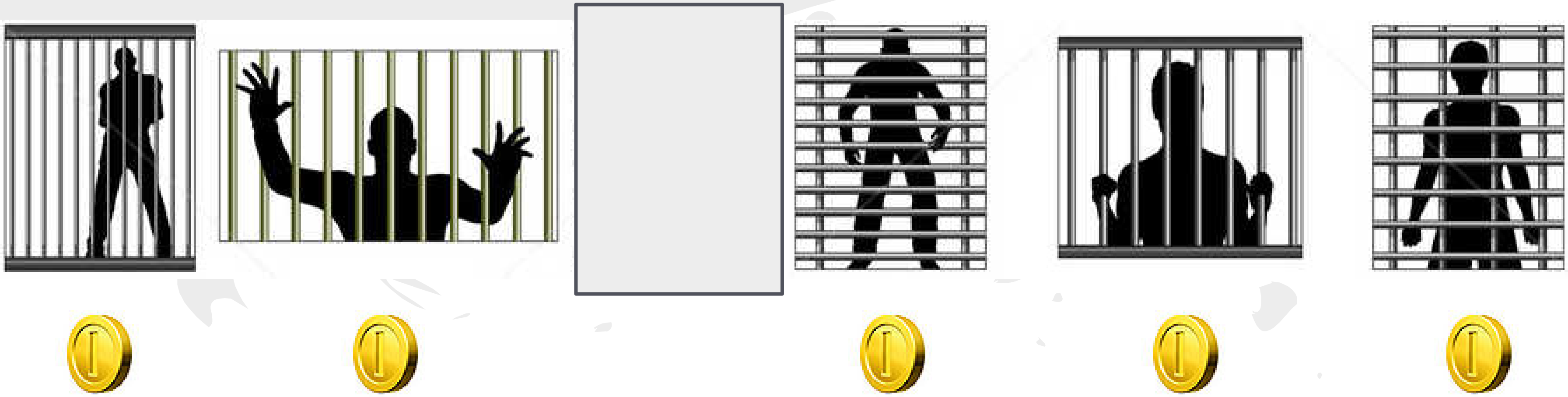
Bribe the Prisoners



Bribe the Prisoners



Bribe the Prisoners



One gold for every prisoner!!

Bribe the Prisoners

- For an array with many prisoners to be freed what is the best order to free them to minimize expenses?

Bribe the Prisoners

- Dynamic Programming saves the day!
  - (and money...)
- For each pair of cells a ≤ b, we want to compute dp[a][b], the best answer if we only have prisoners in cells from a to b, inclusive. Once we decide the location of c, the first prisoner between a and b to be released, we face the smaller sub-problems dp[a][c-1] and dp[c+1][b]. The final answer we want is dp [1][P].

Bribe the Prisoners

```
int p[200]; // prisoners to be released.
map<pair<int, int>, int> dp;

// Finds the minimum amount of gold needed,
// if we only consider the cells from a to b, inclusive.
int Solve(int a, int b) {
  // First, look up the cache to see if the
  // result is computed before.
  pair<int, int> pr(a, b);
  if(mp.find(pr) != mp.end()) return mp[pr];

  // Start the computation.
  int r = 0;
  for(int i=0; i<Q; i++) {
    if(p[i] >= a && p[i] <= b) {
      int tmp = (b-a) + Solve(a, p[i]-1) + Solve(p[i]+1, b);
      if (!r || tmp<r) r=tmp;
    }
  }
  mp[pr]=r;
  return r;
}
```