# Grover's Algorithm

*EE599-001 & EE699-010, Spring 2026*

**Hank Dietz**

http://aggregate.org/hankd/

University of Kentucky

# Unstructured Search

- An associative memory allows lookup by key
  - There is an unordered array of key values
  - Find the entry associated with a given key
  - The value returned is either the entry index or indication that the key was not found
- Unfortunately, superpositions are not arrays
- Implement mapping of the array index → key as a function, which is not always easy

# Unstructured (?) Search

- To find a specific key value, assume that we have a function f() that returns 1 for an index that would map index → *the desired* key
- Assume function f() is an efficient circuit
  - Not a "promise" because f() doesn't need to have a specific structure
  - Might be hard to find such an f()
  - Does conventionally creating f() find the key?

# Can't I Just Reverse Execute?

- If you have a function built entirely using reversible logic, you can find the input that generates a given output by reverse execution
  - Doesn't require a quantum computer
  - Cost is a single evaluation
- Only works if
  - There is just a single x such that f(x)=y
  - There are no ancilla with unknown values

# Quantum Unstructured Search

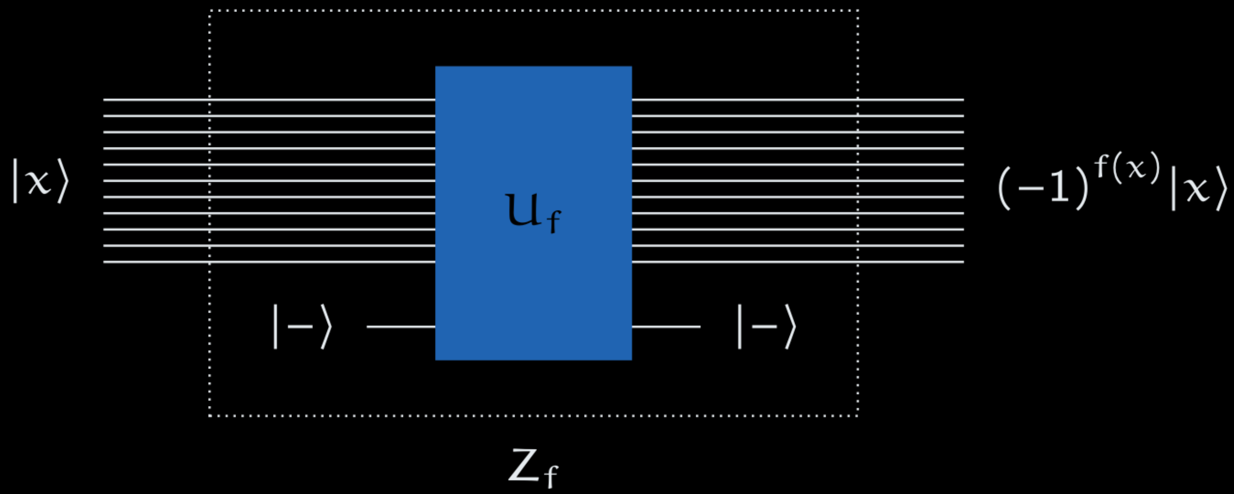
- Given an efficient circuit implementing $f(x) \rightarrow \{0,1\}$, find a value y for which $f(y) \rightarrow 1$
- Output is either
  - y
  - indication that no solution exists

# Complexity for *n*-bit x

- Conventional algorithms
  - Deterministic: $2^n$ evaluations of f()
  - Probabilistic: low probability if less than $O(2^n)$
- Grover's quantum algorithm is $O(\sqrt{(2^n)})$, which is $O(2^{n/2})$

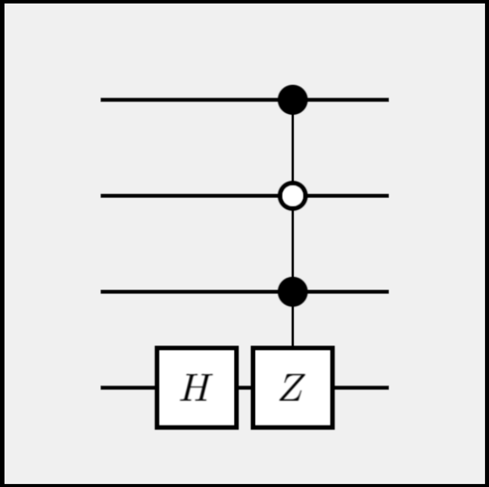
# Create A Phase Query Gate

- Convert f(x) into a unitary function
  $U_f: |a>|x> \rightarrow |a \wedge f(x)>|x>$
- **Phase Query Gate** for f() is
  $Z_f: |x> \rightarrow (-1)^{f(x)}|x>$



# Example Phase Query Gate

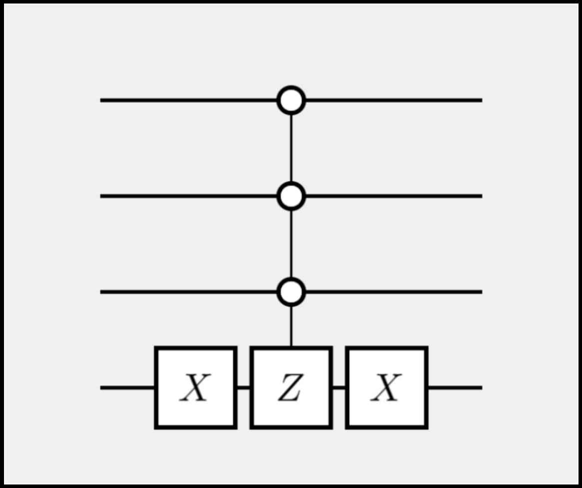
- Suppose f(x) is 1 only when x is 101
- Phase Query Gate for f() is




- Of course, f(x) need not be this obvious…

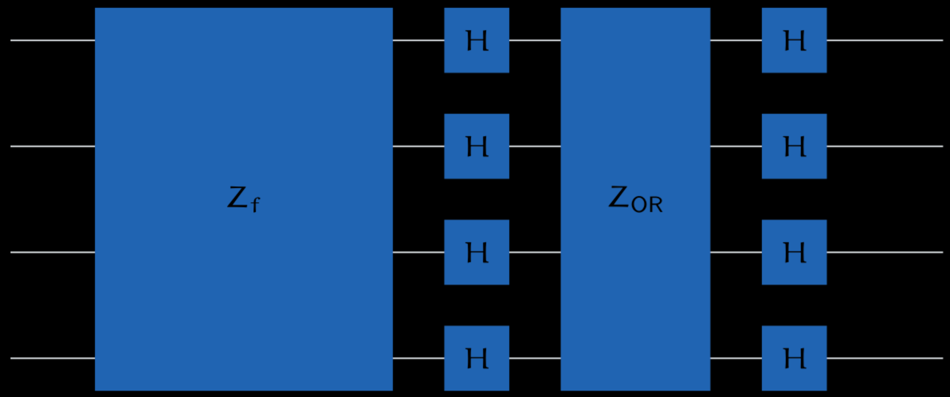
# Need Phase Query *n*-bit OR

- Controlled Z is controlled by AND...
  a OR b is NOT ((NOT a) AND (NOT B))
- Phase Query Gate for OR is



# Grover's Algorithm

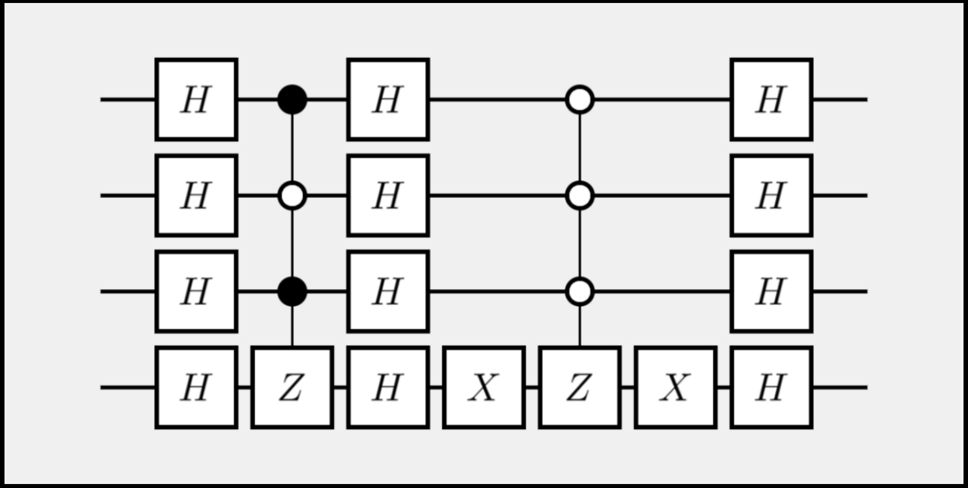
- Initialize n-bit inputs to H|0>
- Apply the Grover operation one or more times:
  $G = H\, Z_{OR}\, H\, Z_f$




- Measure a candidate solution

# Grover's Algorithm Example

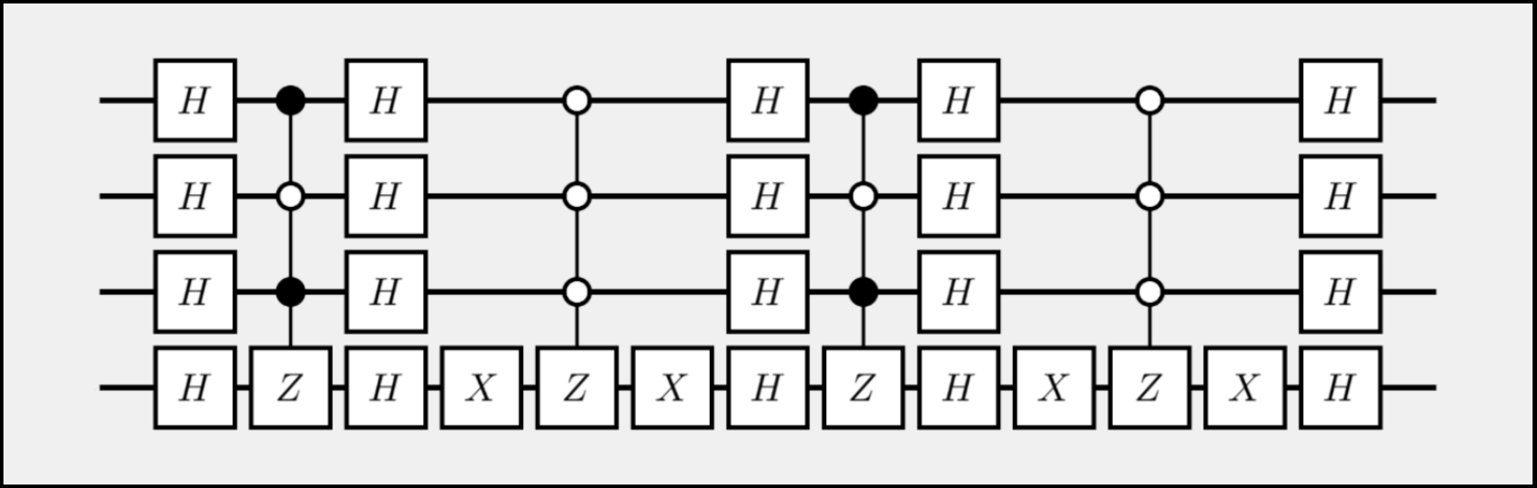
- Suppose f(x) is 1 only when x is 101
- Applying the Grover operation ***once***:




- **Measure a candidate solution**

# Grover’s Algorithm Example

- Suppose f(x) is 1 only when x is 101
- Applying the Grover operation ***twice***:




- **Measure a candidate solution**

# Grover's Algorithm Example

- Usually expressed as functions of $\mathbf{N = 2^n}$
- Rotations reinforce the probabilities
  $\boldsymbol{\theta} = \sin^{-1}(\sqrt{(1/2^n)}) = \sin^{-1}(2^{-n/2}) \approx \mathbf{2^{-n/2}}$
- For a unique single solution, the number of useful repeats of G is $t \approx \text{floor}((\pi/4)\sqrt{(2^n)})$, which is $\mathbf{t \approx floor(\pi 2^{n/2-2})}$
- Probability of success for a unique value is $\mathbf{p(n,1) = \sin^2((2t+1)\,\theta)}$ in t applications of G

# Grover's Algorithm Example

- Probability of success for a unique value is $p(n,1) = \sin^2((2t+1)\,\theta)$ in t applications of G

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $2^n$ | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 1M |
| t | 1.11 | 1.57 | 2.22 | 3.14 | 4.44 | 6.28 | 8.88 | 12.56 | 17.77 | 25.13 | 804.2 |
| p | 0.5 | 0.94 | 0.90 | 0.89 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 |

- $O(2^{n/2})$ values for t is still exponential in *n*

# *s* Solutions

- Usually expressed as functions of $N = 2^n$
- Rotations reinforce the probabilities
  $\theta = \sin^{-1}(\sqrt{(s/2^n)})$ and $t = \text{floor}(\pi/(4\theta))$
- Probability of success for *s* values is
  $p(n,s) \geq \max(1-s2^{-n}, s2^{-n})$

  If you prefer: $p(N,s) \geq \max(1-s/N, s/N)$

# Grover's Algorithm Example

- Usually expressed as functions of $N = 2^n$
- Rotations reinforce the probabilities
  $\theta = \sin^{-1}(\sqrt{(1/2^n)}) = \sin^{-1}(2^{-n/2}) \approx 2^{-n/2}$
- For a unique single solution, the number of useful repeats of G is $t \approx \lfloor(\pi/4)\sqrt{(2^n)}\rfloor$, which is $t \approx \lfloor \pi 2^{n/2-2} \rfloor$
- Probability of success for a unique value is $p(n,1) = \sin^2((2t+1)\,\theta)$ in t applications of G

# Unknown Number of Solutions

- Worst case is still **$O(2^{n/2})$**
- Choose *random* $t \in \{1, . . . , \lfloor \pi N/4 \rfloor\}$
  Probability $\geq$ 40% finding solution (if exists)
- Alternative approach:
  1. Set $T = 1$
  2. Apply G with *random* $t \in \{1, . . . , T\}$
  3. Stop if solution found or timeout with no solution; otherwise, $T = \lceil 1.25T \rceil$ and go to 2.

# Why Grover's Algorithm?

- It can be shown to be **asymptotically optimal**
- It can be applied to many problems
- This technique can be generalized to amplify probabilities for solutions in other problems… this is important because it gives some control over which superposed value is measured
- An excellent mathy explanation is at
  https://quantum.cloud.ibm.com/learning/en/courses/fundamentals-of-quantum-algorithms/grover-algorithm/introduction