Hamiltonian Paths & bitDP

Natalie Bogda & Clara Nguyen

COSC 581 - 04/04/2019

Questions

- What is a Hamiltonian Path?
- What does bitDP stand for?
- What is the time complexity for finding a Hamiltonian Path via DFS? What about via the Held-Karp Algorithm?

About Us

Clara Nguyen

- Master’s Student on Course-Only track.
- Did undergrad at UTK
- Friends with Greg
- Hobbies:
  - Video Games
  - Coding!
  - Music Production
- Born here! Look outside a window for a picture if you want.

Natalie Bogda

- Master’s Student on thesis track.
  - Focus on Computer Vision
- Did undergrad at UTK
- HATES GREG
- Lived in Knoxville since 2011
- Hobbies:
  - Drawing
  - Hiking
  - 50cc 2 stroke scooters

Outline

- What are Hamiltonian Paths?
- The Problem – RainbowGraph
- DFS – The Naïve Approach
- Held-Karp – The Clever Approach via bitDP
- bitDP – Can we go even faster?
- Discussion

What are Hamiltonian Paths?

Hamiltonian Paths

- A path on a graph that visits each vertex exactly once.
- Finding these is an NP-complete problem.

The Problem - RainbowGraph

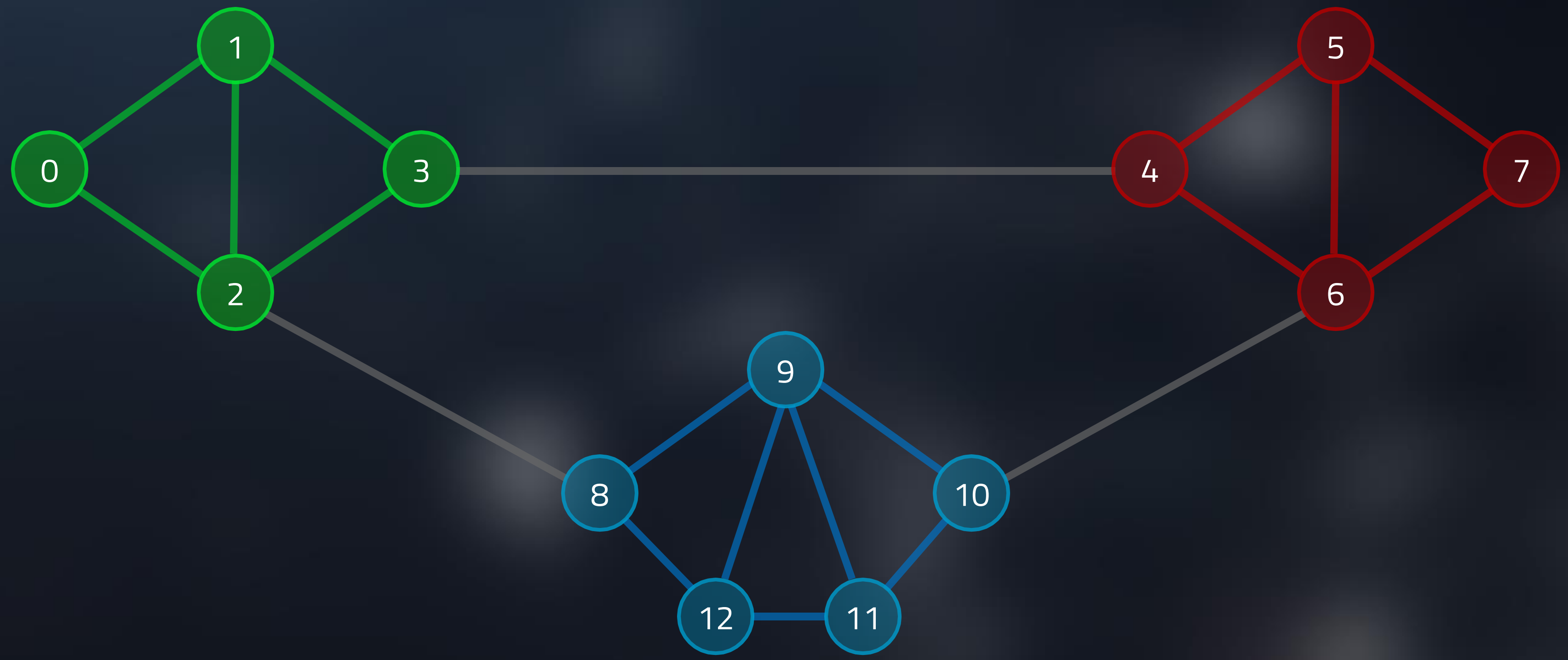
The Problem - RainbowGraph

- Topcoder Problem (SRM 720, D2, 1000-Pointer)
- Find number of Hamiltonian Paths in entire graph
  - Have to count such paths between every possible vertex.
  - via Naïve algorithms, this can *easily* hit $O(N!)$ time.

- Each vertex has a color. If you visit one vertex of a specific color, you have to visit *all* vertices of the same color before going to another.

The Problem - RainbowGraph

- How many Hamiltonian Paths can you find?



DFS – The Naïve Approach

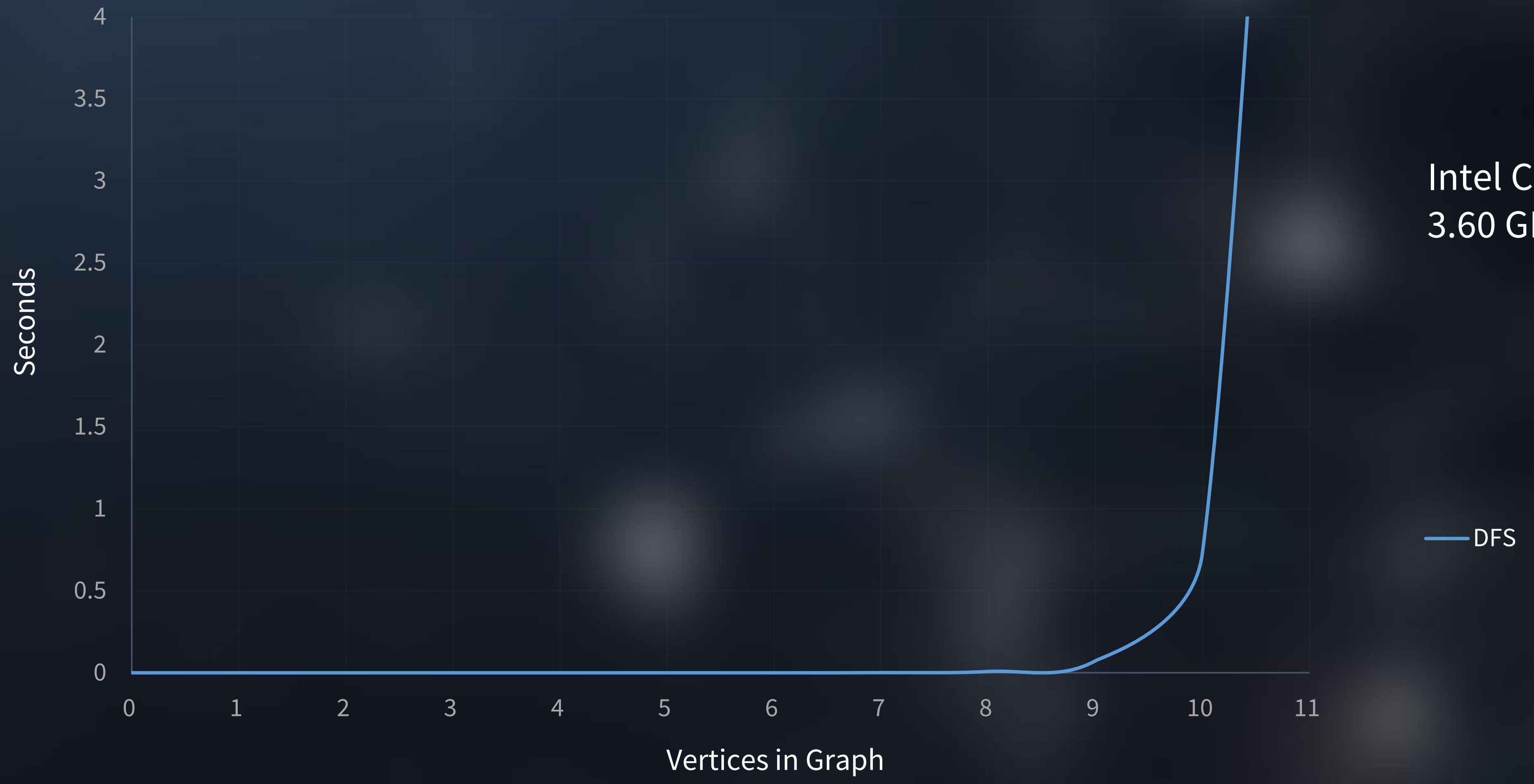
Depth First Search

```
function dfs(a) {
    visited[a] = true;

    if we visited all nodes in graph,
        return true;

    for b is 0 to n - 1
        if (visited[b] == false)
            dfs(b);

    // backtrack
    visited[a] = false;

    return false;
}
```

Depth First Search

- Very naïve way to approach this problem.
- DFS solves the problem in $O(N!)$ time
  - Visit all permutations of vertices in the graph
  - Each iteration, it will traverse the permutation to see if adjacent vertices are connected
  - Therefore $O(N \times N!) = O(N!)$

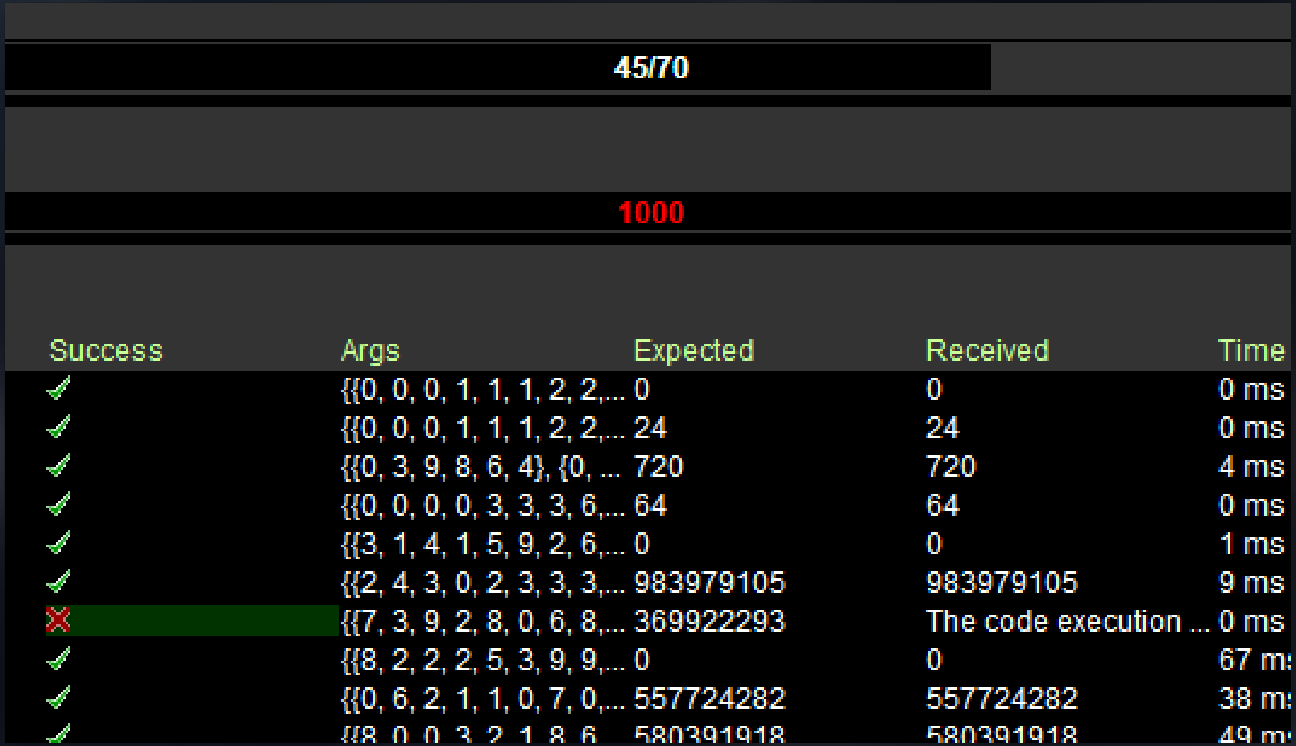
Depth First Search Performance



Intel Core i7-7700
3.60 GHz

DFS on Topcoder (RainbowGraph)

- 45/70 Test Cases complete.
- Too slow!



Held-Karp – The Clever Approach via bitDP

DFS, can we improve it?

- Problems:
  - Multiple repeated function calls.
  - Have to check whether we visited a vertex or not.
  - Recursion.
  - There are properties of these graphs we aren't taking advantage of.
- We have to be clever.

DFS, can we improve it?

- Dynamic Programming? Memoization?
  - Many sub-problems and their results can be cached for later.
  - Reduces the problem down significantly.
- If A can go to B, than B can go to A.
- Use Adjacency Matrix for $O(1)$ lookups. Use Adjacency List for iteration.
- Eliminate recursion as much as possible.
  - Can we get rid of recursion entirely?

Introducing the Held-Karp Algorithm

- Dynamic programming approach developed by Richard Bellman, Michael Held, and Richard Karp in 1962.
  - Solves "sub-problems" to speed up more expensive traversals.
  - Determining if a path exists from A to B becomes $O(N^2)$.
- Reduces DFS's $O(N!)$ time to $O(2^N \times N^2)$.
- How can we implement Held-Karp?

Introducing the Held-Karp Algorithm – Cont.

- **Observe:** If we can go from 0 to 1, and 1 can go to 2, then there is a path involving all three vertices.

- Solve all smaller problems first.

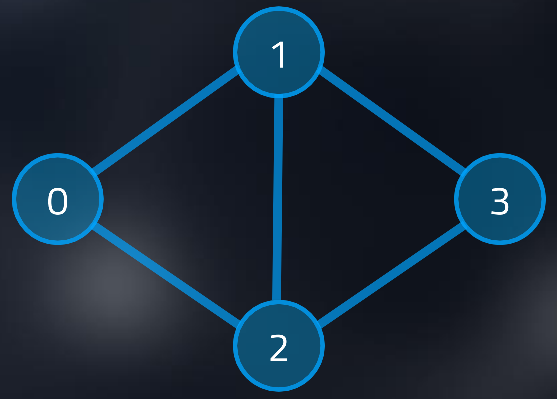
- If it’s possible for all nodes to be visited, there’s a Hamiltonian Path!

bitDP

- bitDP = **Bit D**ynamic **P**rogramming (ビット動的計画法)
- DP table where **vertices** go on one side and **bitmasks** go on the other.
  - Bitmask requires storing all possible combinations of N vertices in bits.
- Table is sized $N \times 2^N$.
  - e.g. Graph with 4 vertices has 16 masks, from 0000 to 1111.
- At the final mask (e.g. 1111), if **any** value is set to 1, there is a Hamiltonian Path in the graph!

bitDP – Example (Reading the table)

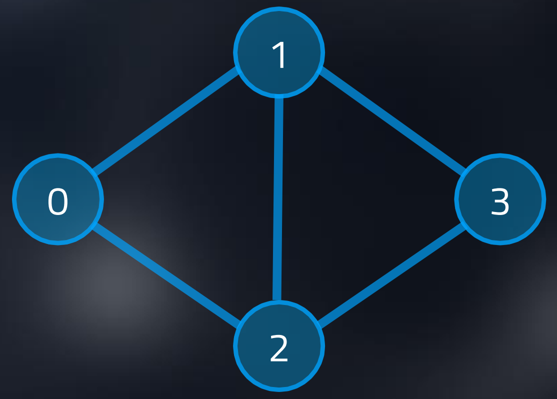
- Make a bitDP table based on the graph:



	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

bitDP – Example (Reading the table)

- Make a bitDP table based on the graph:



	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1
1	0	0	1	1	0	0	1	1	0	0	1	0	0	0	1	1
2	0	0	0	0	1	1	1	1	0	0	0	0	1	0	1	1
3	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1

bitDP – Example (Reading the table)

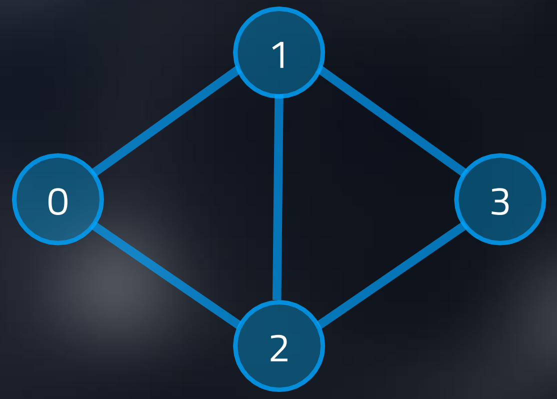
- Consider Mask at 0xB (1011):
  - Vertices Visited: 0, 1, 3

1001	1010	1011	1100	1101
9	A	B	C	D
0	0	1	0	1
0	1	0	0	0
0	0	0	1	0
0	1	1	1	1

- Is there a path between those three that:
  - Ends at 0? Yes
  - Ends at 1? No
  - Ends at 3? Yes

bitDP – Example (Making the table)

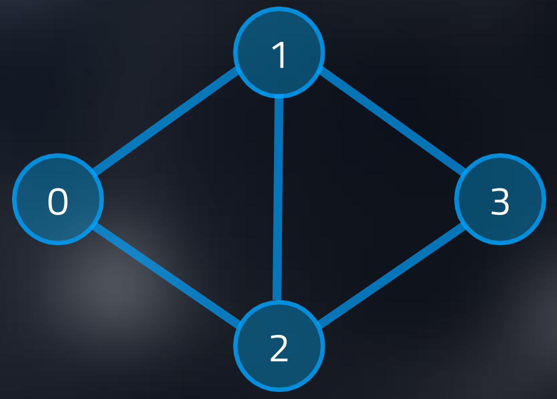
- Make a bitDP table based on the graph:



	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

bitDP – Example (Making the table)

- Each mask only containing 1 vertex is valid.
- These are the simplest “sub-problems”.



	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

bitDP – Example (Making the table)

- Start going through all masks with 2 or more “1”s
- Let’s take a look at 0x3 (0011)… Vertices 0 and 1.
- Go through all rows in column and process.

0001	0010	0011	0100	0101
1	2	3	4	5
1	0	1	0	0
0	1	0	0	0
0	0	0	1	0
0	0	0	0	0

- Column 3, row 0.
  - Compute new mask: 0011 XOR 0001 = 0010
  - Go to mask 0010 and see if any vertex there can go to 0.
  - We can go from vertex 1 to vertex 0. Set the cell to 1.

bitDP – Example (Making the table)

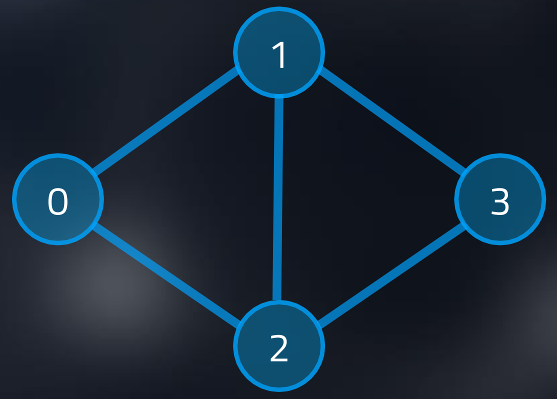
- Start going through all masks with 2 or more “1”s
- Let’s take a look at 0x3 (0011)… Vertices 0 and 1.
- Go through all rows in column and process.

0001	0010	0011	0100	0101
1	2	3	4	5
1	0	1	0	0
0	1	1	0	0
0	0	0	1	0
0	0	0	0	0

- Column 3, row 1.
  - Compute new mask: 0011 XOR 0010 = 0001
  - Go to mask 0001 and see if any vertex there can go to 1.
  - We can go from vertex 0 to vertex 1. Set the cell to 1.

bitDP – Example (Making the table)

- Look at mask 1111… There is a Hamiltonian Path that ends at vertices 0, 1, 2, and 3!



	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1
1	0	0	1	1	0	0	1	1	0	0	1	0	0	0	1	1
2	0	0	0	0	1	1	1	1	0	0	0	0	1	0	1	1
3	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1

bitDP – A few problems

- RainbowGraph doesn't care if we can determine **if a Hamiltonian Path exists**. It cares about **how many** there are.

- We can get how many Hamiltonian Paths **end** with a specific vertex, but we don't know where such paths **started**.

- There is an easy fix... if we bump up the time complexity up a bit.

bitDP – Modifications for Success

- Recall how we set all masks as the first step.
- We can force the grid to give us a starting vertex in a path… by having more tables.

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

bitDP – Modifications for Success

- Split up so each bitmask of 1 vertex gets its own table.

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

bitDP – Modifications for Success

- Once split, simply run the same algorithm again on **all** tables.

dp[0] =

Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

dp[1] =

Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

dp[2] =

Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

dp[3] =

Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

bitDP – Modifications for Success

- Once split, simply run the same algorithm again on **all** tables.
- Now we can determine if a Hamiltonian Path exists from a **start** and **end** vertex.

dp[0] =

Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1

dp[1] =

Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1

dp[2] =

Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0
2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

dp[3] =

Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1
2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

bitDP – Modifications for Success

- Modify the algorithm to **add** to a cell, rather than set it to 1.

- We now get the **total** number of Hamiltonian paths!

dp[0] =

Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	2

dp[1] =

Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1

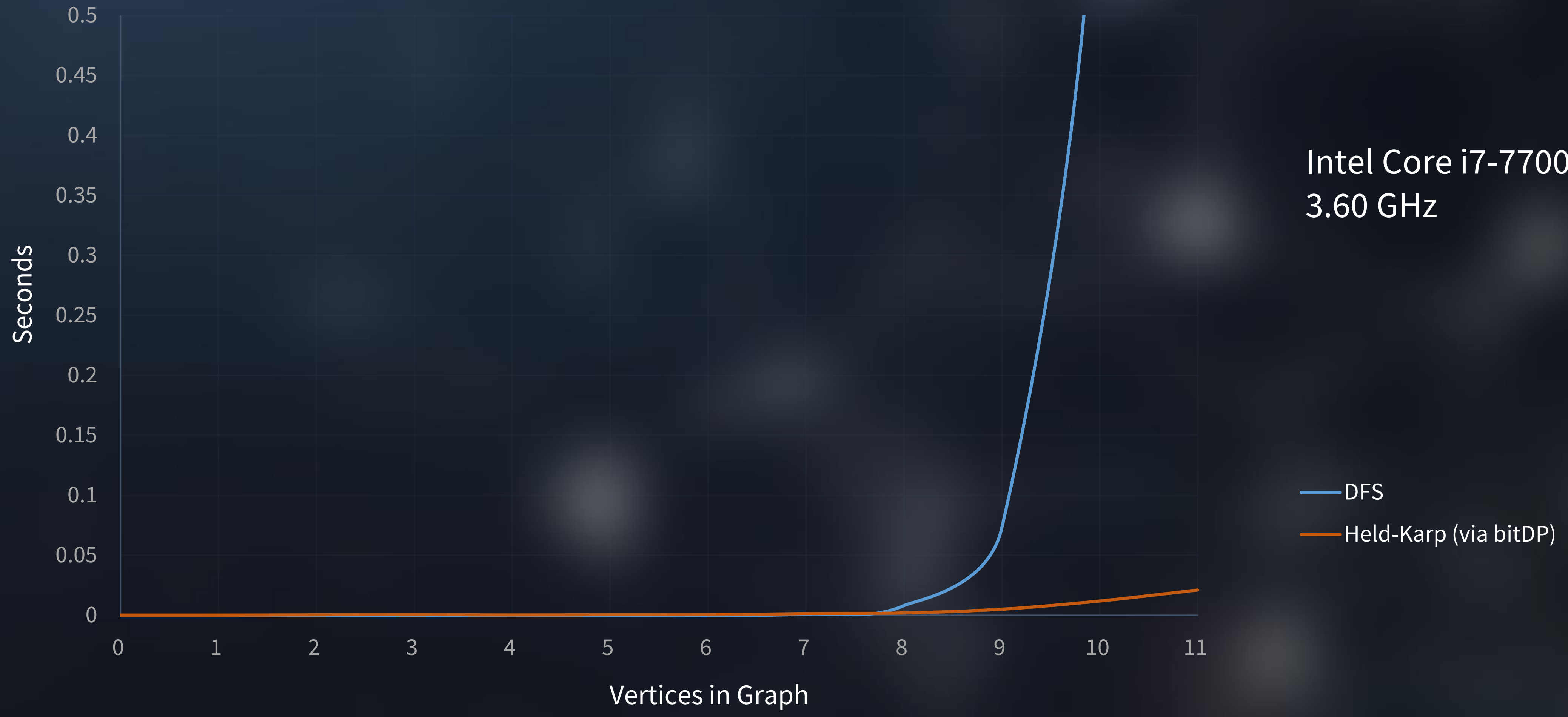
dp[2] =

Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0
2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

dp[3] =

Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	2
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1
2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

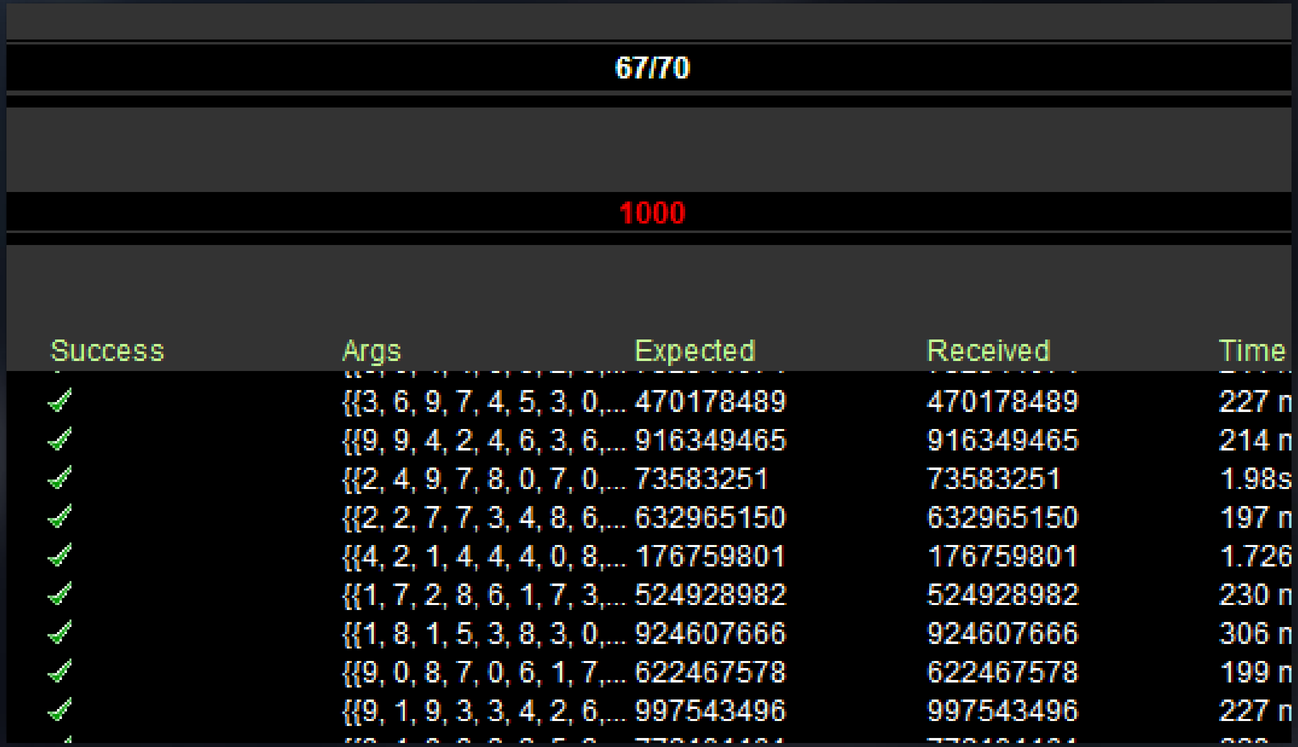
Held-Karp (via bitDP) Performance



Intel Core i7-7700
3.60 GHz

bitDP on Topcoder (RainbowGraph)

- 67/70 Test Cases complete.
- Still too slow!



67/70

1000

Success	Args	Expected	Received	Time
✓	{{3, 6, 9, 7, 4, 5, 3, 0,...	470178489	470178489	227 n
✓	{{9, 9, 4, 2, 4, 6, 3, 6,...	916349465	916349465	214 n
✓	{{2, 4, 9, 7, 8, 0, 7, 0,...	73583251	73583251	1.98s
✓	{{2, 2, 7, 7, 3, 4, 8, 6,...	632965150	632965150	197 n
✓	{{4, 2, 1, 4, 4, 4, 0, 8,...	176759801	176759801	1.726
✓	{{1, 7, 2, 8, 6, 1, 7, 3,...	524928982	524928982	230 n
✓	{{1, 8, 1, 5, 3, 8, 3, 0,...	924607666	924607666	306 n
✓	{{9, 0, 8, 7, 0, 6, 1, 7,...	622467578	622467578	199 n
✓	{{9, 1, 9, 3, 3, 4, 2, 6,...	997543496	997543496	227 n

Blue Flame Moonshine
128
Ole Smoky
Ole Smoky Tennessee Moonshine
BLUE FLAME MOONSHINE
64% ALC./VOL (128 PROOF)
OLE SMOKY DISTILLERY
Gatlinburg Tennessee
SPIRITS DISTILLED FROM GRAIN AND CANE
GOVERNMENT WARNING:
HEALTH PROBLEMS.


JACK DANIEL'S
Old No.7 BRAND
Tennessee
SOUR MASH
WHISKEY
DISTILLED & BOTTLED BY
JACK DANIEL DISTILLERY
LYNCHBURG, TENN. USA
40% ALC. BY VOL. (80 PROOF)
QUALITY & CRAFTSMANSHIP SINCE 1866

bitDP (Alcohol-Induced) on Topcoder (RainbowGraph)

- 70/70 Test Cases complete.
- Barely passes

70/70

1000

Success	Args	Expected	Received	Time
✓	{{0, 0, 0, 1, 1, 1, 2, 2,...	0	0	0 ms
✓	{{0, 0, 0, 1, 1, 1, 2, 2,...	24	24	0 ms
✓	{{0, 3, 9, 8, 6, 4}, {0, ...	720	720	3 ms
✓	{{0, 0, 0, 0, 3, 3, 3, 6,...	64	64	0 ms
✓	{{3, 1, 4, 1, 5, 9, 2, 6,...	0	0	1 ms
✓	{{2, 4, 3, 0, 2, 3, 3, 3,...	983979105	983979105	14 m
✓	{{7, 3, 9, 2, 8, 0, 6, 8,...	369922293	369922293	218 n
✓	{{8, 2, 2, 2, 5, 3, 9, 9,...	0	0	61 m
✓	{{0, 6, 2, 1, 1, 0, 7, 0,...	557724282	557724282	191 n
✓	{{8, 0, 0, 3, 2, 1, 8, 6,...	580391918	580391918	46 m

bitDP – Can we go faster?

Speeding bitDP up

- Skip computations we know won't work:
  - Skip all "from" vertices not set to "1" in a mask.
  - Skip all "to" vertices not set to "1" in a mask.
  - Skip Column 0 as it is never used.
- If A can go to B, then B can go to A.
- All cells before a vertex in a row are guaranteed to be 0.

Speeding bitDP up – Part 1

- If A can go to B, then B can go to A.
- Symmetry exists between DP table rows:

dp[1] =

Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1

dp[2] =

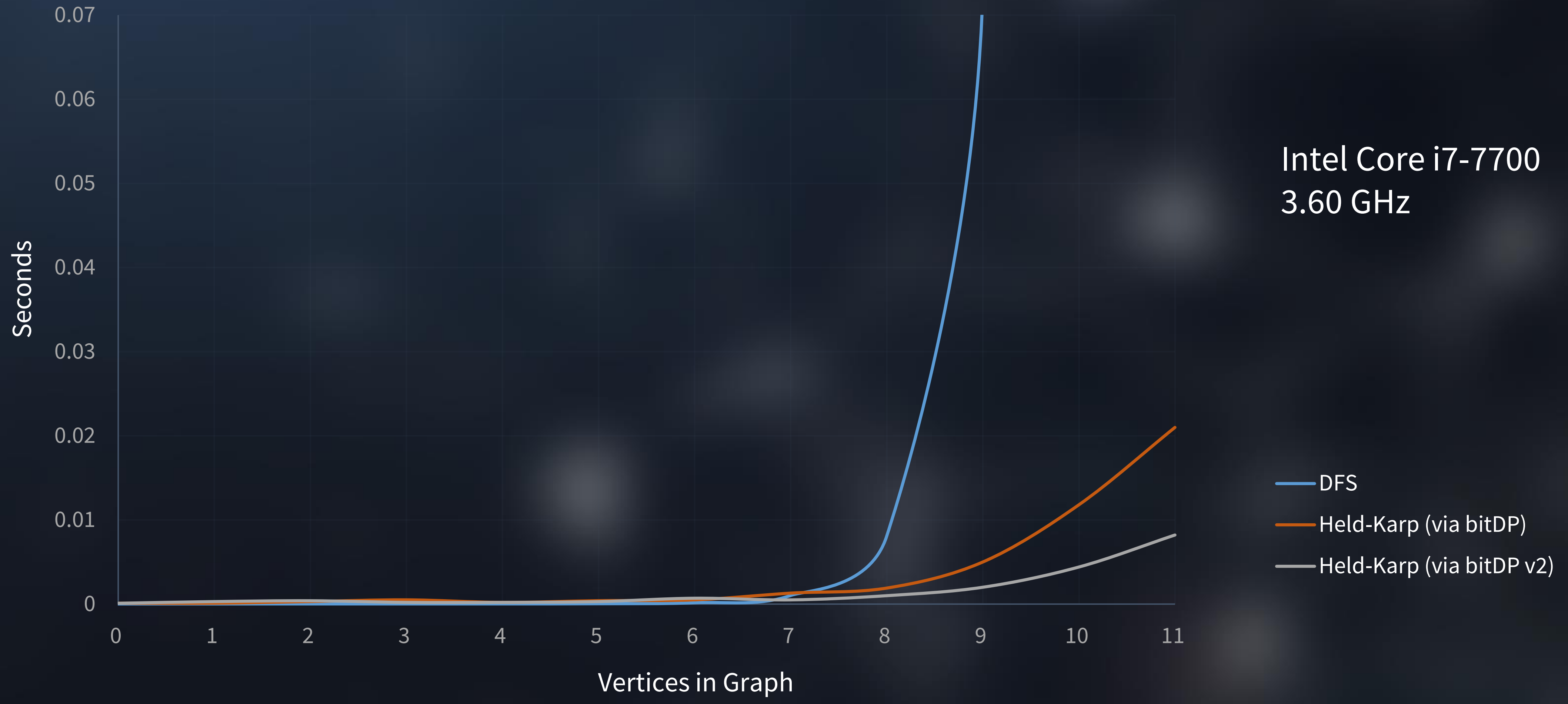
Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0
2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

Speeding bitDP up – Part 2

- All cells before a vertex in a row are guaranteed to be 0.

Vertex/Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1
1	0	0	1	1	0	0	1	1	0	0	1	0	0	0	1	1
2	0	0	0	0	1	1	1	1	0	0	0	0	1	0	1	1
3	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1

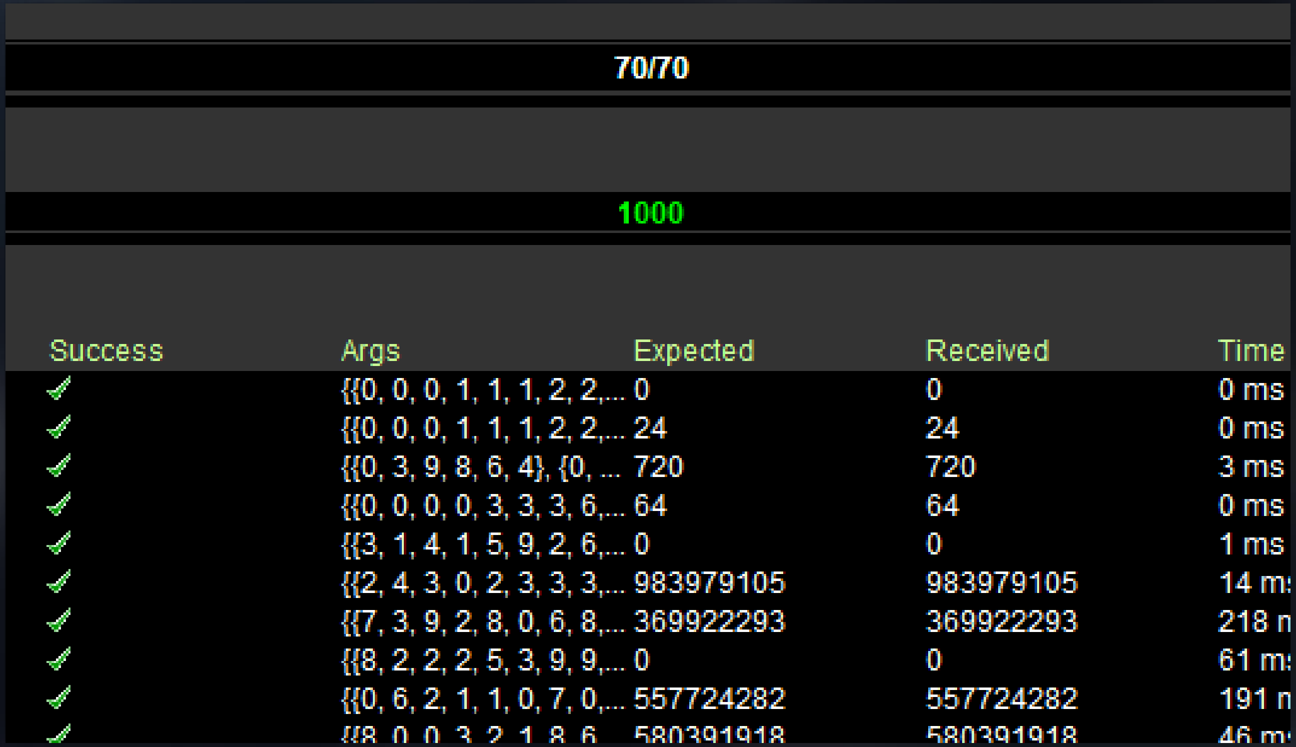
Held-Karp (via bitDP v2) Performance



Intel Core i7-7700
3.60 GHz

bitDP v2 on Topcoder (RainbowGraph)

- 70/70 Test Cases complete.
- Barely passes



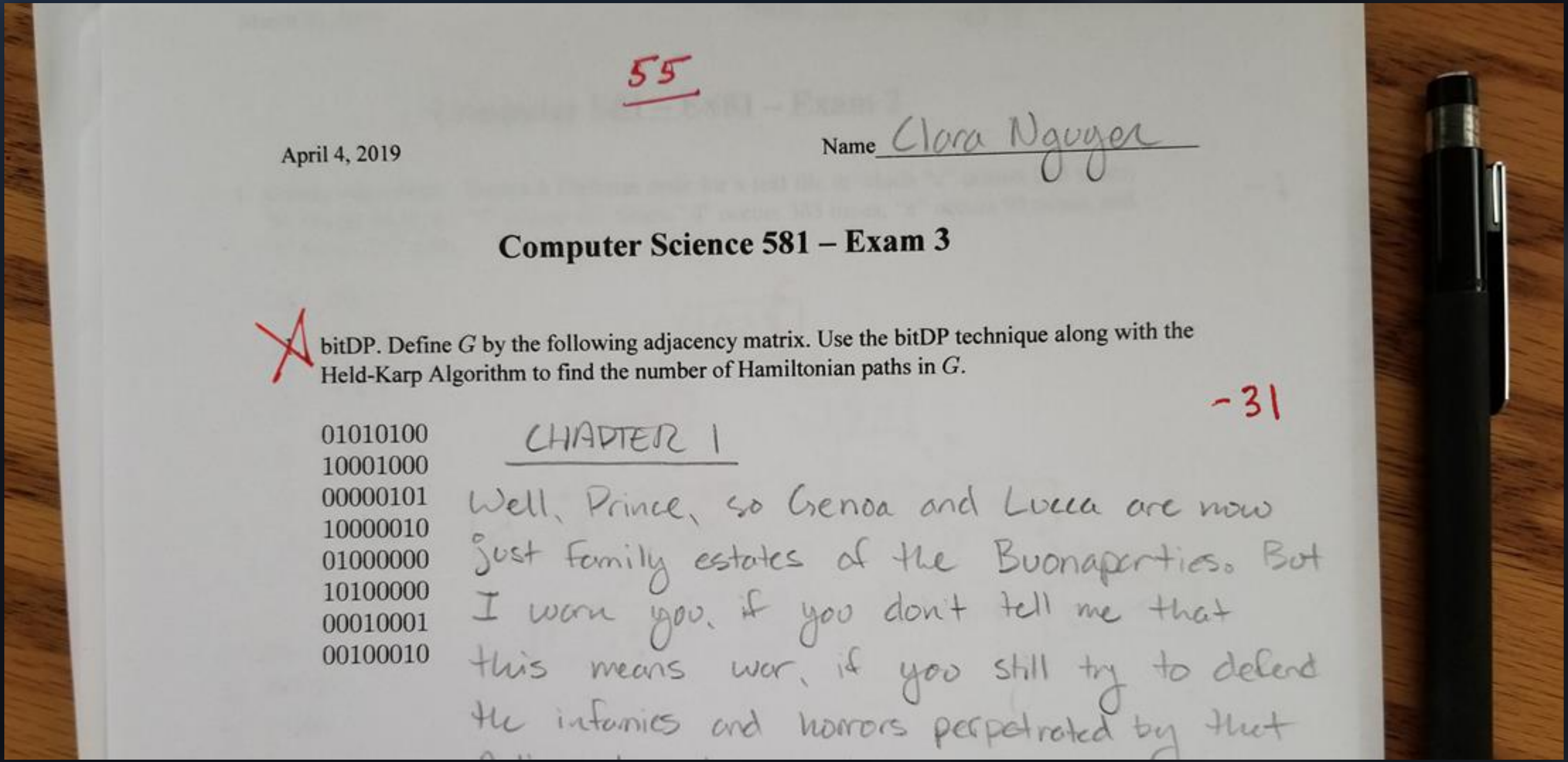
70/70

1000

Success	Args	Expected	Received	Time
✓	{{0, 0, 0, 1, 1, 1, 2, 2,...	0	0	0 ms
✓	{{0, 0, 0, 1, 1, 1, 2, 2,...	24	24	0 ms
✓	{{0, 3, 9, 8, 6, 4}, {0, ...	720	720	3 ms
✓	{{0, 0, 0, 0, 3, 3, 3, 6,...	64	64	0 ms
✓	{{3, 1, 4, 1, 5, 9, 2, 6,...	0	0	1 ms
✓	{{2, 4, 3, 0, 2, 3, 3, 3,...	983979105	983979105	14 m
✓	{{7, 3, 9, 2, 8, 0, 6, 8,...	369922293	369922293	218 m
✓	{{8, 2, 2, 2, 5, 3, 9, 9,...	0	0	61 m
✓	{{0, 6, 2, 1, 1, 0, 7, 0,...	557724282	557724282	191 m
✓	{{8, 0, 0, 3, 2, 1, 8, 6,...	580391918	580391918	46 m

One more thing…

Leaked Exam 3



Discussion

Questions

- What is a Hamiltonian Path?
- What does bitDP stand for?
- What is the time complexity for finding a Hamiltonian Path via DFS? What about via the Held-Karp Algorithm?
- BONUS: Who is Greg?

References

- AtCoder Inc. “実践・最強最速のアルゴリズム勉強会 第四回講義資料(ワークスアプリケーションズ & AtCoder).” LinkedIn SlideShare, 29 Mar. 2014, www.slideshare.net/chokudai/wap-atcoder4.
- “BitDP.” *CCS 千葉大学電子計算機研究会*, 13 Mar. 2019, densanken.com/wiki/index.php?BitDP.
- “Held–Karp Algorithm.” *Wikipedia*, Wikimedia Foundation, 19 Feb. 2019, en.wikipedia.org/wiki/Held%E2%80%93Karp_algorithm.
- Jaimini, Vaibhav. “Hamiltonian Path Tutorials & Notes | Algorithms.” *HackerEarth*, www.hackerearth.com/ja/practice/algorithms/graphs/hamiltonian-path/tutorial/.
- Nguyen, Clara. “RainbowGraph: A Better Approach.” *RainbowGraph: A Better Approach*, 25 Jan. 2019, utk.claranguyen.me/guide.php?id=rainbowgraph_bitdp.
- Plank, James S. “SRM 720, D2, 1000-Pointer (RainbowGraph).” *CS494 Lab 6*, 3 Dec. 2018, 15:25, web.eecs.utk.edu/~plank/plank/classes/cs494/494/labs/Lab-6-RainbowGraph/.
- “Problem Statement for RainbowGraph.” *TopCoder Statistics - Problem Statement*, community.topcoder.com/stat?c=problem_statement&pm=14667.

Hamiltonian Paths & bitDP

Natalie Bogda & Clara Nguyen

COSC 581 - 04/04/2019