Problem Solving:
Paradigms & Applications

Paul S. Wang, Sofpower.com

May 20, 2023

One of the main purposes of computational thinking (CT) is to better solve problems. Some would even say CT is basically a problem-solving methodology.

Having seen many interesting applications of CT for problem solving in earlier articles, we are now ready to look at "*problem solving*" itself in general.

Figure 1: Thinking about Problem Solving

This post is part of our *Computational Thinking* (CT) blog where you can find many other interesting and useful articles.

Problem Solving Paradigms

We shall briefly explain some important general problem solving paradigms, methods, and approaches and give examples.

- **Decomposition** or **Divide and Conquer**: This involves breaking down a large or complex problem into smaller, more manageable parts or sub-problems. This approach allows you to focus on one part of the problem at a time and solve it before moving on to the next.

  Example: To create a website, you can decompose the task into smaller tasks such as designing the layout, writing the HTML code, adding CSS styling, creating interactive features, and running user tests.

- **Abstraction**: This involves identifying the essential features of a problem and ignoring the irrelevant details (Figure 2). Abstraction allows you to simplify a problem and focus on its core components.

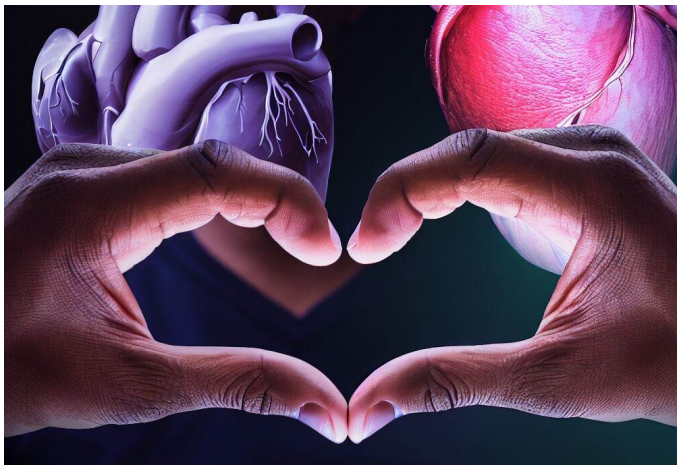

Figure 2: Abstract Heart Gesture

  Example: To create a game, you can abstract away the complexities of the physics engine and focus on the game mechanics and gameplay.

- **Algorithmic Thinking**: This involves developing step-by-step procedures or algorithms for solving problems. Algorithms provide a clear and unambiguous set of instructions that can be followed to solve a problem.

Example: To find the shortest path between two points on a map, you can use Dijkstra's algorithm which involves assigning a tentative distance to each node and updating the distance as you move along the graph.

- **Pattern Recognition**: This involves identifying patterns (Figure 3) or trends in data that can be used to make predictions or solve problems. Pattern recognition allows you to identify similarities and differences between different data sets.

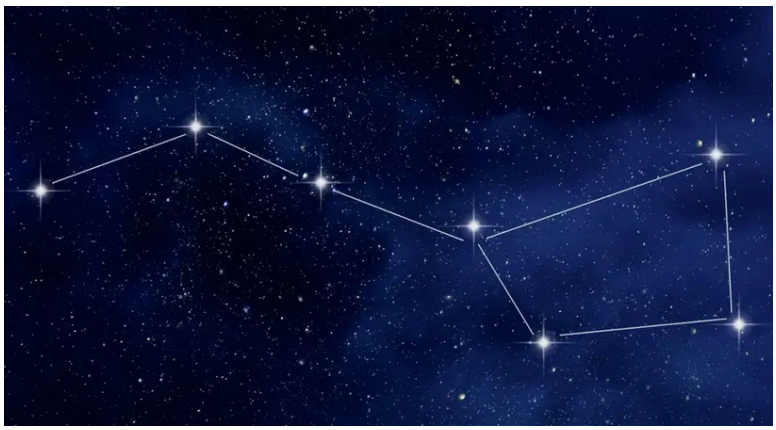

Figure 3: Big Dipper in the Night Sky

Example: To identify fraudulent credit card transactions, you can use pattern recognition to identify common characteristics of fraudulent transactions such as unusual purchase amounts or locations.

- **Logical Reasoning**: This involves using deductive reasoning to analyze and solve problems. Logical reasoning involves identifying the premises of an argument and drawing conclusions based on those premises.

  Example: To solve a Sudoku puzzle, you can use logical reasoning to eliminate numbers from each cell based on the numbers already present in the same row, column, and 3x3 box.

- **Top-Down Design**: This approach involves starting with a broad view of a problem and then breaking it down into smaller, more specific sub-problems. It allows you to focus on the high-level structure of a problem before diving into the details.

Example: To plan a dinner party, you might start with the number of guests, the food (dishes and drinks), any dietary considerations of individual guests, and the order of presentation of the courses, and so on.

- **Bottom-Up Design**: This approach involves starting with specific details and building them up into a larger goal. It is useful when dealing with problems where the details are well-defined but the overall structure is flexible.

  Example: Again consider the dinner party. We can start with the needed ingredients, cleaning them, preparing each ingredient to be ready for cooking, then cook each required dishes in a given order. Finally the whole dinner is served as planned.

- **Recursion**: This involves solving a problem by breaking it down into smaller, similar sub-problems that can be solved using the same method. It is useful for solving problems that have a recursive structure. We talked about recursion in part one. Here is a mirror image version of recursion (Figure 4).

Figure 4: Infinity Mirror Demonstrating Recursion

  Example: To calculate the factorial of a number, you can use a recursive function that repeatedly multiplies the number by itself minus one until the base case of 1 is reached.

- **Iteration**: This involves repeating a set of steps or a process multiple times until a desired outcome is achieved. It is useful for refining solutions and improving their quality.

Example: To develop a machine learning model, you might iterate through several rounds of training and testing to gradually improve its accuracy and performance.

- **Heuristics**: This involves using rules of thumb or educated guesses to solve a problem when an optimal, systematic, or algorithmic solution is too slow, too hard, or not feasible. It is useful when dealing with complex problems where finding an optimal solution is computationally infeasible.

  Example: Consider security checking at crowded public places such as train stations or airports. *Profiling*–looking for suspicious actions or behaviors in the crowd–is often effective. Other heuristics include common sense, stereotyping, educated guess, and even intuition.

- **Backtracking**: This involves systematically exploring all possible solutions to a problem by trying one option after another and backtracking when a dead-end is reached. It is useful for solving problems with a large search space.

  Example: For maze (Figure 5) escapting, you can use backtracking to explore all possible paths until the correct path is found.

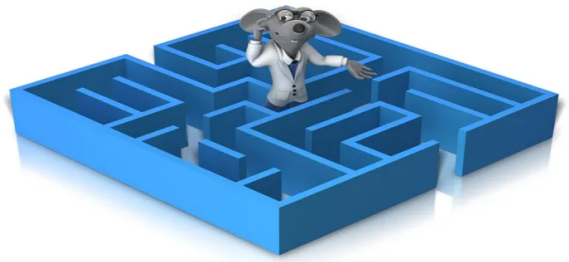

Figure 5: Part of A Garden Maze

- **Trial and Error**: This involves trying out different solutions until a desired outcome is achieved. It is useful when dealing with problems where the solution is not immediately clear.

  Example: To solve a crossword puzzle, you can use trial and error to try out different word combinations until the correct solution is found.

- **Incremental Improvements**: This involves making small changes to a solution to gradually improve its performance or quality. It is useful when dealing with problems where the solution can be refined through small changes.

  Example: To improve a software program, you can use incremental improvements to add new features, fix bugs, and optimize its performance over time, resulting in *version 2.0*.

- **Parallelization**: This involves breaking a problem down into smaller sub-problems and solving each sub-problem simultaneously on multiple processors (Figure 6) or cores. It is useful for solving problems that are computationally intensive and can be broken down into smaller, independent tasks.

Figure 6: Working in Sync

  Example: To look for someone or something in an area, we can divide the area into a grid and employ multiple searchers ssimultaneously, each responsible for a subarea.

- **Optimization**: This involves finding better or even the best possible solution to a problem within a given set of constraints. It is useful for solving problems where there are many possible solutions, but only a few of them are optimal.

  Example: To optimize the production schedule for a factory, you can use mathematical optimization techniques such as linear programming to find the schedule that maximizes production while satisfying constraints such as labor availability and machine capacity.

- **Modularity**: This involves breaking a system down into smaller, independent modules or components that can be developed and tested separately. It is useful for solving problems that are too large or complex to be solved as a whole.

  Example: To run a restaurant, you can divide up the operations into reservations, hosting, table service, kitchen, bar service, and so on.

- **Structuring Data**: Solution methods can become easier, faster, and more efficient if the data involved in the problem or solution process are organized to have certain structures that the solution method can utilize to advantage.

  Example: A dictionary of words is an obvious example. Records kept by time and date, airline flights by departure and destination cities, addresses by city, state, and zipcode, files and folders in a hierarchical (tree) structure are additinal examples.

Of course there are many other CT paradigms, methods, and approaches used in problem solving. By using a combination of these techniques, you can solve complex problems in a structured and effective manner.

In fact, the more familiar you are with computational and digital technologies the better you can solve all kinds of problems.

Applications in Daily Living

Let's see how useful the CT problem solving paradigms are in our daily lives.

- **Planning a trip**: When planning a trip (Figure 7), you can use decomposition to break down the trip into smaller tasks such as booking flights, reserving accommodations, and planning activities. You can also use heuristics to find the best time to book flights and accommodations based on historical pricing data, and trial and error to find the best deals. Additionally, you can use optimization to find the best itinerary that maximizes your time and minimizes your expenses.

- **Solving household problems**: When facing household problems, such as a leaky faucet or a malfunctioning appliance, you can use decomposition to break down the problem into smaller components and

Figure 7: Trip Planning

identify the root cause. You can also use trial and error to try out different solutions until you find one that works, and incremental improvements to optimize the solution over time.

- **Managing personal finances**: When managing your personal finances, you can use data abstraction to simplify your budget and identify areas where you can cut expenses. You can also use optimization to find the best investment strategies that maximize your returns while minimizing your risks.

- **Choosing a career**: When choosing a career, you can use top-down design to identify your career goals and then break them down into smaller steps such as researching job opportunities and acquiring relevant skills. You can also use data abstraction to simplify the job market and identify the most promising career paths, and heuristics to identify industries and job roles that match your interests and skills.

- **Learning a new skill**: When learning a new skill, you can use bottom-up design to start with the basics and gradually build up your knowledge and expertise. You can also use iteration to practice your skills and refine your technique over time, and feedback loops to identify areas where you need to improve.

- **Making decisions**: When making decisions, you can use logical reasoning to analyze the pros and cons of different options, and heuristics to identify biases and errors in your decision-making process. You can

also use backtracking to explore different options and find the best solution, and trial and error to test different strategies until you find the one that works best for you.

Handling Crises

It is true that no one can be lucky all the time. When something bad happens, such as a loved one had an accident or your kitchen caught on fire, how do you avoid panic so you don't get into more trouble yourself. There is a reason why people say "bad things come in pairs." This is indeed a problem worth solving ahead of time.

Let's see how CT problem solving techniques can help us through a crisis. Here are some tips for dealing with emergencies:

1. **Stay calm**: In an emergency situation, it's natural to feel panicked or overwhelmed. However, it's important to try to stay calm and focused in order to make effective decisions and take correct actions (Figure 8).

Figure 8: Stay Calm Can Save Lives

2. **Call for help**: If the situation requires immediate medical attention, call emergency services or seek help from a nearby hospital or clinic. If the situation involves a safety threat, call the police or other appropriate authorities.

3. **Take immediate action**: If there is a specific action that you can take to address the situation, take it as soon as possible. For example, if someone is choking, perform the Heimlich maneuver immediately.

4. **Follow emergency protocols**: If you are in a workplace or public setting, follow the emergency protocols that are in place. These protocols may include evacuation procedures, emergency contacts, and other important information.

5. **Communicate with others**: If there are others involved in the situation, communicate with them clearly and calmly. Try to provide reassurance and guidance as needed.

6. **Seek support**: After the immediate crisis has passed, seek support from friends, family members, or professionals if needed. It's important to take care of yourself and seek help if you are struggling with the emotional impact of the situation.

Normally, it's important to be prepared and take a structured approach to problem-solving. But, the priority in an emergency is to take immediate action to deal with the urgent present problems. By staying calm, taking action, and seeking help as needed, you can help to minimize the impact of the emergency and protect yourself and others.

In the End

In our three articles on CT problem solving, we have discussed many topics and hopefully gained a much better understanding and become better at solving problems of many kinds.

However, in addition to the paradigms, methods, and approaches discussed earlier, here are a few more important aspects of computational thinking for problem-solving—creativity, collaboration, Continuous improvement adapting to change, and ethical considerations.

Computational thinkers, by incorporating these aspects into your problem-solving approach, you can develop more effective solutions and make a positive impact on the world around you.