Problem Solving & Software Development

Lecture 2. Algorithmic Strategies. Brute Force Approach

Problem Solving Approaches

- Talk to the person next to you.
- List three strategies you would use as a problem solving strategy in programming?

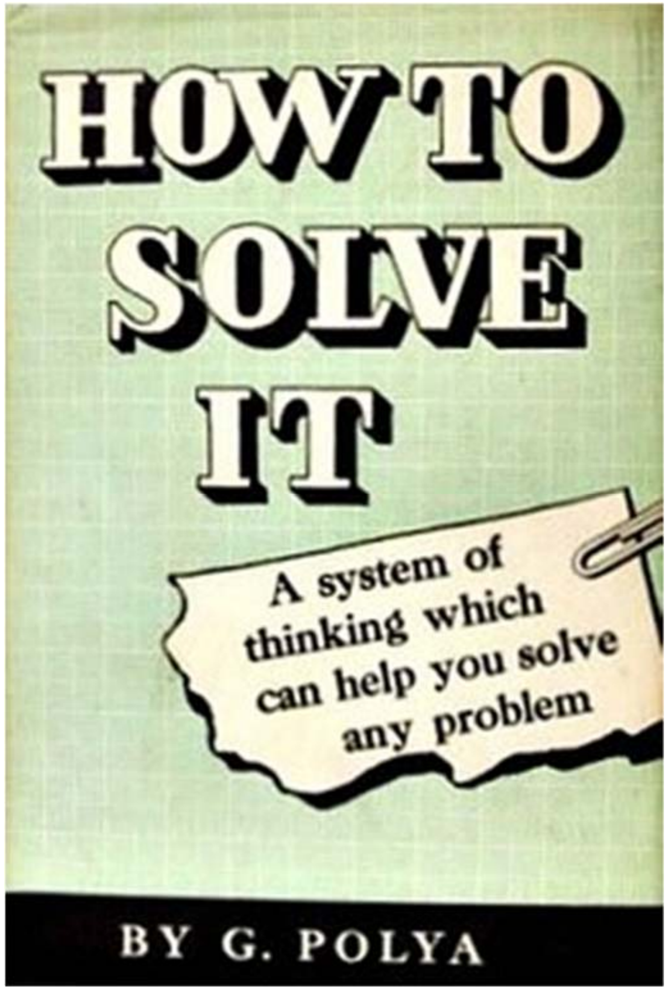
- When back home, read at least https://en.wikipedia.org/wiki/How_to_Solve_It

Problem Solving Approaches

- First, you have to understand the problem.
- After understanding, then make a plan.
- Carry out the plan.
- Look back on your work. How could it be better?

Possible Problem Solving Strategies:

- Guess and check
- Make an orderly list
- Eliminate possibilities
- Use symmetry
- Consider special cases
- Solve an equation
- Look for a pattern
- Draw a picture
- Solve a simpler problem
- Decompose and Recombine
- etc.



Algorithmic Strategies

Problems can be classified by their solution techniques.

Some problems may fit into more than one category.

- Brute force approach
- Recursive approach
- Dynamic programming
- Greedy algorithms
- Divide and Conquer
- Geometric approach
- Algorithms on graphs

Brute Force Approach (Exhaustive Search)

Systematically enumerate all possible candidates for the solution and check whether each of them satisfies the problem's statement.

Use when

- the simplicity of implementation is more important than speed;
- the problem size is limited;
- the best we can do is consider all possible solutions.

The cost of the approach is proportional to the number of candidate solutions.
=> Try to speed up the brute-force search for your problem.

Brute Force Approach (Exhaustive Search)

Examples of problems:

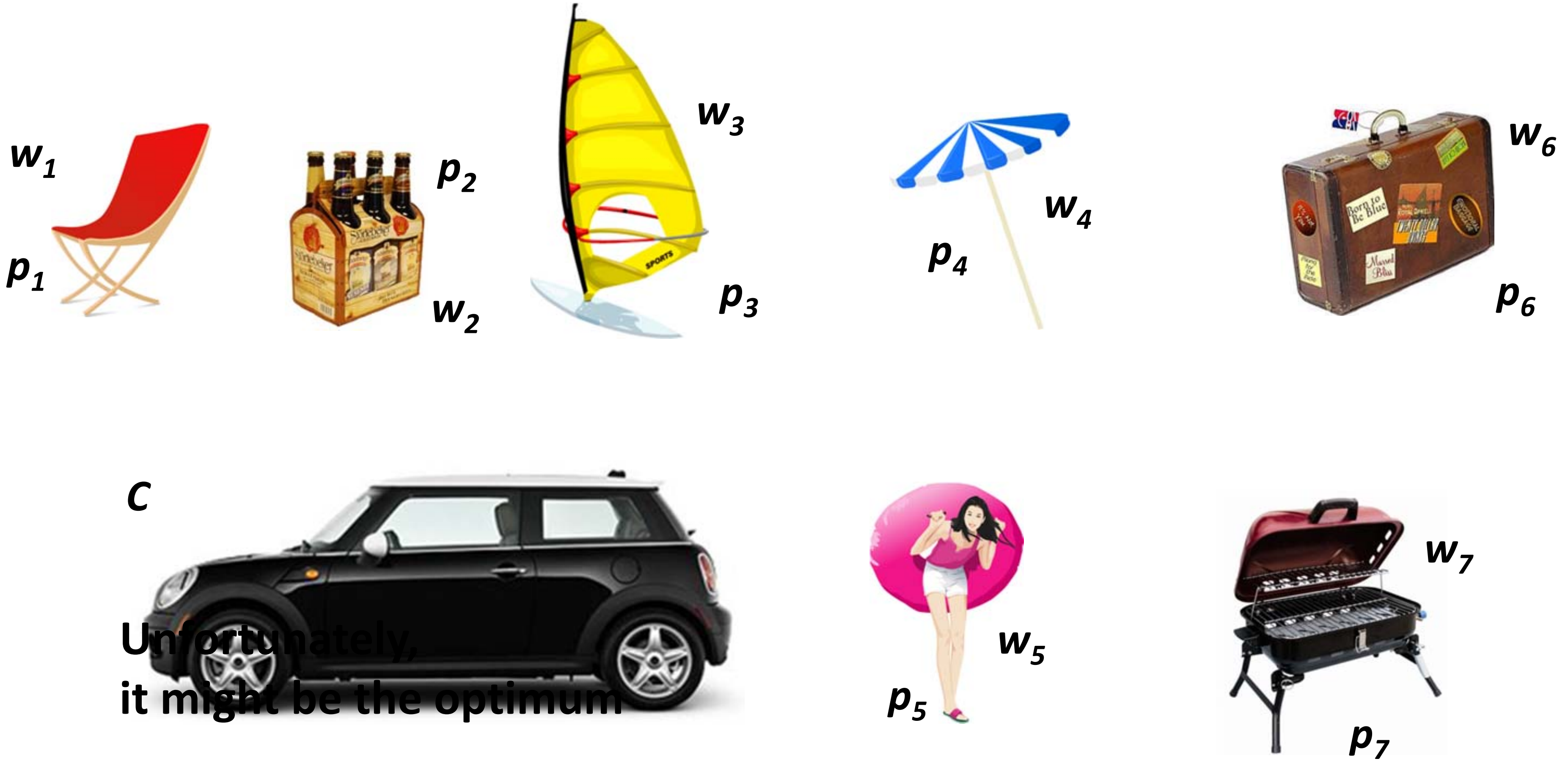
- Searching in unsorted data
- Travelling Salesman Problem
- Enumerating Permutations
- Knapsack problem
- Closest pair of points problem

Knapsack Problem

Preparing for a vacation? Solve the knapsack problem first!

The sum of weights of selected elements must not exceed the capacity C.

Maximise the total profit of collected elements .



Unfortunately,
it might be the optimum

Knapsack Problem

Brute Force Approach (Exhaustive Search)

Closest pair of points problem

Find the closest pair of points in a 2-d space.

Input: An array $[(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)]$

Output: Two points: (x_i,y_i) (x_j,y_j)

Brute Force Approach (Exhaustive Search)

Closest pair of points problem

Find the closest pair of points in a 2-d space.

Input: An array $[(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)]$

Output: Two points: (x_i,y_i) (x_j,y_j)

```
minDist = infinity
for i = 1 to n-1
   for j = i + 1 to n
       if dist(i, j) < minDist:
           minDist = dist(i, j)
           closestPair = (i, j)
return closestPair
```

Greedy Algorithms

Make the locally optimal choice at each stage (with the hope) to find a global optimum.

- Greedy algorithms mostly fail to find the globally optimal solution, because they usually do not operate exhaustively on all the data.
- If a greedy algorithm can be proven to yield the global optimum for a given problem, it typically becomes the method of choice.

Examples of problems:

- Giving change (with Australian currency)
- Huffman Encoding
- Finding minimum spanning trees (Kruskal's algorithm and Prim's algorithm)

Recursive approach

Recursion?

Recursive approach

Emphasis of iteration:

keep repeating until a task is “done”.

Emphasis of recursion:

break the problem up into smaller parts; combine the results.

Which is better?

- Recursive solutions are often shorter.
- Iterative solutions keep control local to loop, less “magical”
- Good recursive solutions may be more difficult to design and test.

Recursive approach

Recursive is not always better!

```
// Fibonacci: recursive version
int Fibonacci_R(int n) {
   if(n<=0) return 0;
   else if(n==1) return 1;
   else return Fibonacci_R(n-1)+Fibonacci_R(n-2);
}
```

This takes $O(2^n)$ steps! Unusable for large n.

```
// Fibonacci: iterative version
int Fibonacci_I(int n) {
   int fib[] = {0,1,1};
   for(int i=2; i<=n; i++) {
      fib[i%3] = fib[(i-1)%3] + fib[(i-2)%3];
   }
   return fib[n%3];
}
```

This iterative approach is “linear”; it takes $O(n)$ steps.

Divide and Conquer Paradigm

Recursively break down a problem into sub-problems of the same or related type, until these become simple enough to be solved directly.

The solutions to the sub-problems are then combined to give a solution to the original problem.

Examples of problems:

- Sorting data (Quick Sort, Merge Sort)
- Integer mutiplication (School method, Karatsuba algorithm)
- Closest pair of points problem

Dynamic Programming

- Solve a complex problem by breaking it down into a collection of simpler sub-problems.
- Solve each of those sub-problems just once, and store their solutions.
- The next time the same sub-problem occurs,
  instead of recomputing its solution, one simply look up
  the previously computed solution.

This saves computation time at the expense of a (hopefully) modest expenditure in storage space.

Examples of problems:

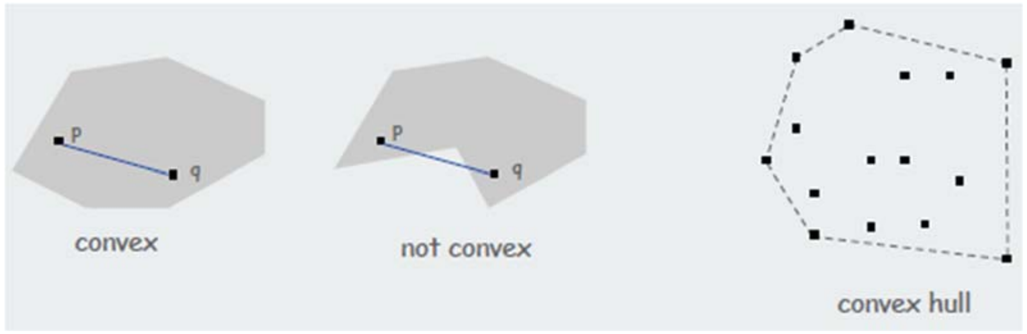
- Minimimum edit distance
- Shortest path problem (Dijkstra’s or Floyd–Warshall algorithms)

Geometric Algorithms

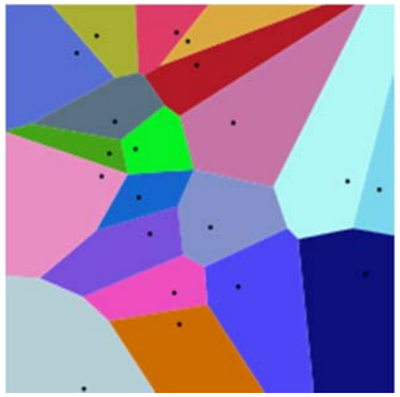
Determine geometric aspects in your problem.

Examples of problems:

- Closest pair of points problem
- Convex Hull



- Vornoi Diagram



Geometric Algorithms

A farmer has bought an **110 hectares** piece of land. He has decided to grow **wheat** and **barley** on that land. The entire production can be sold.

He wants to know how to plant each variety in the 110 hectares, given the costs, net profits and labor requirements according to the data shown below:

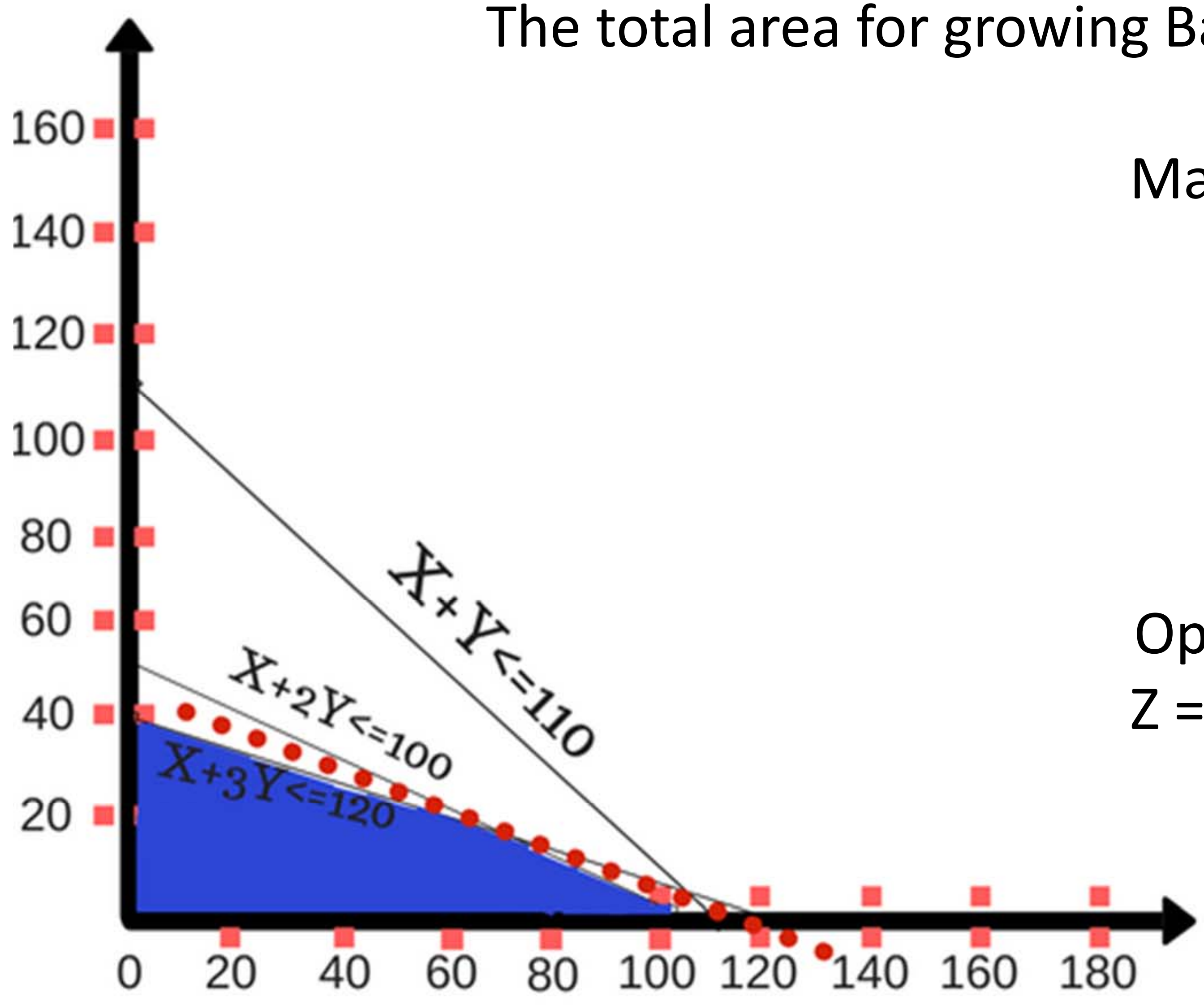
Variety	Cost (Price/Hec)	Net Profit (Price/Hec)	Man-days/Hec
Wheat	100	50	10
Barley	200	120	30

The farmer has a **budget of $10,000** and an availability of **1,200 man-days** during the planning horizon.

Find the best solution and the optimal value.

Geometric Algorithms

The total area for growing Wheat = X (in hectares)
The total area for growing Barley = Y (in hectares)



Maximise $Z = 50X + 120Y$

$100X + 200Y \leq 10{,}000$

$10X + 30Y \leq 1200$

$X + Y \leq 110$

$X \geq 0, Y \geq 0$

Optimum $(X,Y) = (60,20)$.

$Z = 50 * (60) + 120 * (20)$

Algorithms on graphs

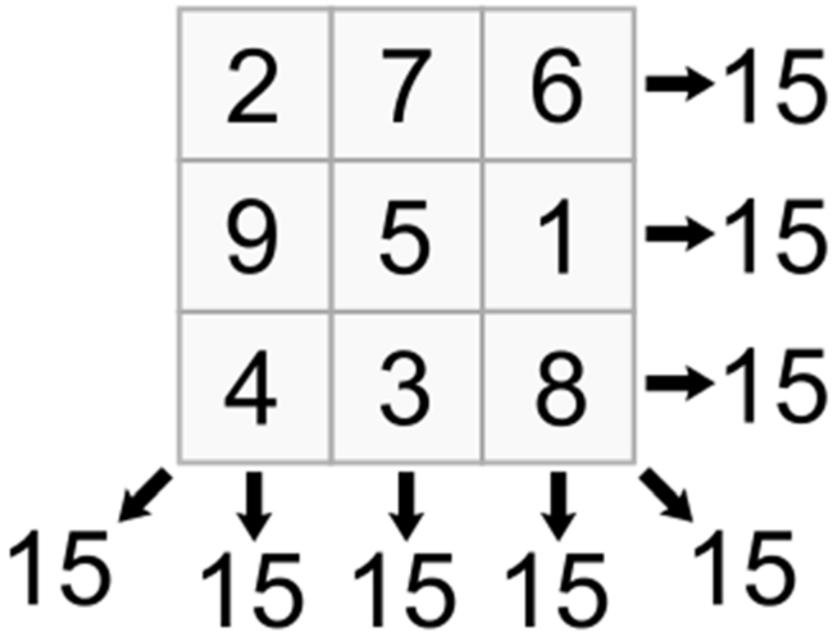
Represent your problem as a graph

Examples of problems:

- Shortest path problem (Dijkstra’s or Floyd–Warshall algorithms)
- Searching for connected components
- Breadth first search (BFS) and depth first search (DFS)
- Maximum weighted bipartite matching

Problem Exercise

- Lottery Ticket
- Generating a 3x3 magic squares



- Bus Seating