Problem Solving Paradigms

Lesson 3

How to solve problems?

- Ad hoc
- Use known algorithms
- Complete search
- Greedy algorithm
- Divide and conquer
- Dynamic programming

Today

Plan

- Paradigms overview
- Usage Examples
- Tips

Complete Search

- A.k.a. brute force, recursive backtracking.
- Strategy: traverse the entire search space.

- When to use?
  - Use when the input is small enough (risk of TLE)

Greedy Solutions

- Strategy: choose the local optimum at each step.
- E.g. Kruskal, Prim

- When to use?
  - When optimal steps result in the optimum.

Divide & Conquer

- Strategy:
  - Divide the problem to smaller subproblems
  - conquer each subproblem
  - Combine solutions
- Examples: merge sort, binary search.

- When to use binary search?
  - o Looking for a minimal value k for which some property holds.
  - o Given such k, it is easy to check if the property holds.
  - o The property is **monotonous**. If the property holds for k, then it holds for all j>k.

Dynamic Programming

- Strategy: Recursion + Re-use

- When to use it?
  - A solution can be constructed efficiently from solutions to subproblems.
  - Some subproblems overlap.

Examples

Simple Equations (UVa 11565)

- Input: integers $1 \leq A, B, C \leq 10^4$
  Find: different integers x, y, z such that
  $$x + y + z = A$$
  $$x \cdot y \cdot z = B$$
  $$x^2 + y^2 + z^2 = C$$
- Solution: complete search.
- Range?
  $$x^2 \leq C \leq 10^4 \;\Rightarrow\; -100 \leq x \leq 100$$
  same for y and z $\quad -100 \leq y \leq 100$
  $$-100 \leq z \leq 100$$

Simple Equations (UVa 11565)

```
bool sol = false; int x, y, z;
for (x = -100; x <= 100; x++)
  for (y = -100; y <= 100; y++)
    for (z = -100; z <= 100; z++)
      if (y != x && z != x && z != y
          && x + y + z == A
          && x * y * z == B
          && x * x + y * y + z * z == C) {
        if (!sol) printf("%d %d %d\n", x, y, z);
        sol = true;
      }
```

Simple Equations (UVa 11565)

```
bool sol = false; int x, y, z;
for (x = -100; x <= 100; x++)
  for (y = -100; y <= 100; y++) if (y != x)
    for (z = -100; z <= 100; z++)
      if (y != x && z != x && z != y
          && x + y + z == A
          && x * y * z == B
          && x * x + y * y + z * z == C) {
        if (!sol) printf("%d %d %d\n", x, y, z);
        sol = true;
      }
```

Simple Equations (UVa 11565)

```
bool sol = false; int x, y, z;
for (x = -100; x <= 100; x++) if (x * x <= C)
  for (y = -100; y <= 100; y++) if (y != x && x * x + y * y <= C)
    for (z = -100; z <= 100; z++)
      if (y != x && z != x && z != y
          && x + y + z == A
          && x * y * z == B
          && x * x + y * y + z * z == C) {
        if (!sol) printf("%d %d %d\n", x, y, z);
        sol = true;
      }
```

Simple Equations (UVa 11565)

```
bool sol = false; int x, y, z;
for (x = -22; x <= 22; x++) if (x * x <= C)
  for (y = -100; y <= 100; y++) if (y != x && x * x + y * y <= C)
    for (z = -100; z <= 100; z++)
      if (~~y != x && ~~z != x && z != y
          && x + y + z == A
          && x * y * z == B
          && x * x + y * y + z * z == C) {
        if (!sol) printf("%d %d %d\n", x, y, z);
        sol = true;
      }
```

Placing Gas Stations

- Goal: Place gas stations along a given road system. Find the least amount of gas stations that will cover the road.
- We only have a “test function”:
  - Given k, the function returns true iff we can cover the road system using k gas stations or less.
- Clearly, if we can consturct k gas stations to cover the road, then we can also do it for every j>k.
- Use binary search to find the minimal possible value of k.

Problem

- You want to buying a car using a loan:
  - The car costs 1000$.
  - The bank charges 10% of the unpaid loan at the end of every month.
  - You want to pay a fixed amount d for 2 months.
  - What is d?

- Solution:
  - D&C (bisection method, look for the answer)
  - Define a reasonable range and binary search it.

Solution Simulation

- Range: $0.01 < d < 1100$
- Step 1: $d = 550?$
  Debt after 1 month: $1000 \cdot 1.1 - d = 550$
  Debt after 2 months: $550 \cdot 1.1 - d = 55 > 0$
  $$d > 550$$
- Step 2: $d = 825?$
  Debt after 1 month: $1000 \cdot 1.1 - d = 275$
  Debt after 2 months: $275 \cdot 1.1 - d = -522.5 < 0$
  $$d < 825$$
- Additional steps…

Coin Change

- Input: target amount V, coin values.
  Output: Min number of coins that form V.

- Example: $V = 17, coins = \{10,5,1\}$
  We need 4 coins (10+5+1+1)

- Idea: Greedy algorithm. Choose the largest coin that doesn't exceed the remaining value.

Coin Change

- Input: target amount V, coin values.
  Output: Min number of coins that form V.

- Example: $V = 6, coins = \{4,3,1\}$
- We need 2 coins (3+3)
  The greedy solution uses 3 coins (4+1+1)

- Idea: DP.
  (Greedy works for certain coin values,
  DP works in general).

Coin Change

- Input: target amount V, coin values.
  Output: Min number of coins that form V.

- Define $Change(v)$: the num of coins needed for V.
- $Change(v) = 1 + \min_{c \in coins} change(v - c)$
- $Change(0) = 0, Change(< 0) = \infty$
- We need $Change(V)$
- Compute $Change(v)$ for $v = 0,1,2 \dots V$

Tips

DP

- Used to compute the value of a recursive function f, assuming:
  - f has no cyclic dependencies
  - Each argument has a finite number of possible value
  - The set of all possible parameter combinations is small enough to fit in memory

Steps for DP

1. Define the subproblem
   (the function meaning and parameters)
2. Find the recursive rule
3. Solve base cases
4. Define the target value
   (which value do you need to solve the problem)
5. Define the computation order
   (e.g. ascending order of n)

Tips for complete search

- How to generate all subsets
  - For the subsets of $\{1, \dots, n\}$ use the binary representation of integers between 0 and $2^n - 1$.

```
for (i = 0; i < (1 << n); i++) {
  // i represents a subset
  for (int j = 0; j < n; j++) if (i & (1 << j)) {
    …
    // j represents an element in i
  }
}
```

Tips for complete search

- How to generate all permutations
  - Use next_permutation from the STL algorithm library.
  - To go over all permutations, start with the sorted.

```
#include <algorithm>
int n = 8, p[8] = {0, 1, 2, 3, 4, 5, 6, 7};
do {
  …
} while (next_permutation(p, p + n));
```

Binary Search: STL

- sort: sort a range of values
- Searching a sorted range of values:
  - Binary search(x): Return true/false if x value exists
  - lower_bound(x): Return a pointer to the first element which is greater or equal to x.
  - upper_bound(x): Return a pointer to the first element which is strictly greater than x.
- For example, when searching for 3 in a sorted array:
  - 1 2 3 3 3 3 4 6
  - 1 2 4 6

Binary Search: integers

```
bool can(int f) { ...}

int lo = 0, hi = 10000, mid = 0, ans = 0;
while (lo < hi) {
  mid = (lo + hi) / 2;
  if (can(mid)) hi = mid;
  else {lo = mid+1; ans = lo;}
}

printf("%d\n", ans);
```

Binary Search: fractions

```
#define EPS 1e-9
bool can(double f) { ...}

double lo = 0.0, hi = 10000.0, mid = 0.0, ans = 0.0;
while (fabs(hi - lo) > EPS) {
  mid = (lo + hi) / 2.0;
  if (can(mid)) {hi = mid; ans = hi}
  else lo = mid;
}

printf("%.3lf\n", ans);
```

Remark: Sorting

- Sorting is a useful tool for a wide range of problems.
- After sorting it is easier to:
  - Quickly search for elements (D&C)
  - Go over the elements in order
  - Find identical elements
- Example: Lawn Mower (first lesson)
- When facing a problem, try to think if ordering the elements might help after some manipulations.