This course material is now made available for public usage.
Special acknowledgement to School of Computing, National University of Singapore
for allowing Steven to prepare and distribute these teaching materials.

CS3233 Competitive Programming

Dr. Steven Halim

Week 03 – Problem Solving Paradigms

(Complete Search; Divide & Conquer; Greedy)

Outline

- Mini Contest 2 + Break + Discuss Last Week's C, Today's A/B
- Admins
- Complete Search
  - Iterative: (Nested) Loops, Permutations, Subsets
  - Recursive Backtracking
  - Some Tips
- Greedy Algorithm (we will discuss this first before D&C)
  - Ingredients and some examples
- Divide and Conquer (D&C)
  - Focus on **Binary Search the Answer**

Iterative: (Nested) Loops, Permutations, Subsets

Recursive Backtracking

Some Tips

COMPLETE SEARCH

Iterative Complete Search (1)

- UVa 725 – Division
  - Find two 5-digits number s.t. → **abcde** / **fghij** = N
  - **abcdefghij** must be all different, 2 <= N <= 79
- Iterative Complete Search Solution (Nested Loops):
  - Try all possible **fghij** (one loop)
  - Obtain **abcde** from **fghij * N**
  - Check if **abcdefghij** are all different (*another* loop)

Iterative Complete Search (2)

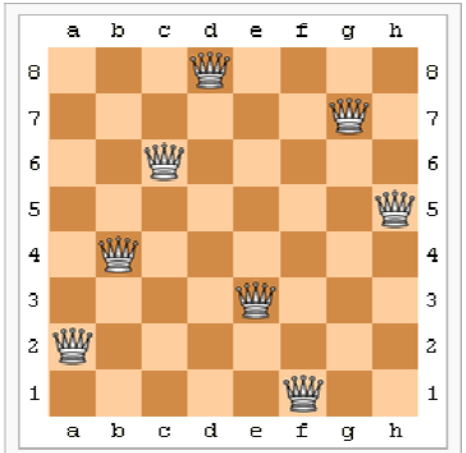
- UVa 11742 – Social Constraints
  - There are $0 < \mathbf{n} \leq 8$ movie goers
  - They will sit in the front row with **n** consecutive open seats
  - There are $0 \leq \mathbf{m} \leq 20$ seating constraints among them, i.e. **a** and **b** must be at most (or at least) **c** seats apart
  - How many possible seating arrangements are there?
- Iterative Complete Search Solution (Permutations):
  - Set counter = 0 and then try all possible **n! permutations**
  - Increase counter if a permutation satisfies all **m** constraints
  - Output the final value of counter

Iterative Complete Search (3)

- UVa 12346 – Water Gate Management
  - A dam has $1 \leq \mathbf{n} \leq 20$ water gates to let out water when necessary, each water gate has **flow rate** and **damage cost**
  - Your task is to manage the opening of the water gates in order to get rid of *at least* the specified **total flow rate** condition that the **total damage cost** is minimized!
- Iterative Complete Search Solution (Subsets):
  - Try all possible $\mathbf{2^n}$ subsets of water gates to be opened
  - For each subset, check if it has sufficient flow rate
    - If it is, check if the total damage cost of this subset is smaller than the overall minimum damage cost so far
      - If it is, update the overall minimum damage cost so far
  - Output the minimum damage cost

Recursive Backtracking (1)

- UVa 750 – 8 Queens Chess Problem
  - Put 8 queens in 8x8 Chessboard
  - No queen can attack other queens
- Naïve way (Time Limit Exceeded)
  - Generate 8^8 = 17M possible 8 queens position and filter the infeasible ones…



One solution.

Recursive Backtracking (2)

- Better way, recursive backtracking
  - Insight 1: **all-different constraint** for row (or col)
    - Search space goes down from 8^8 = 17M to 8! = 40K!
  - Insight 2: diagonal check
    - Another way to prune the search space
    - Queen A (i, j) attacks Queen B (k, l) iff `abs(i - k) == abs(j - l)`
- Scrutinize the sample code of recursive backtracking!

Tips

1. Filtering versus Generating
2. Prune Infeasible Search Space Early
3. Utilize Symmetries
4. Pre-Computation a.k.a. Pre-Calculation
5. Try Solving the Problem Backwards
6. Optimizing Your Source Code
7. Use Better Data Structure & Algorithm ☺

More Search Algorithms…

- Depth Limited Search
- Iterative Depth Limited Search
- A*
- Iterative Deepening A* (IDA*)
- Memory Bounded A*
- Branch and Bound (BnB)
- Maybe in Week12 ☺...

Ingredients and Non Classical Example

GREEDY ALGORITHM

Coin Change (Special Case)

- Given many coins of different denominations = {25, 10, 5, 1} cents, make X cents with the *least number of coins* used!
  - If X = 42, solution: 25+10+**5+1+1** = 42 (5 coins)
  - If X = 17, solution: 10+**5+1+1** = 17 (4 coins)
  - If X = 7, solution: **5+1+1** = 7 (3 coins)
- Observe "optimal sub-structure" and "Greedy property"!

Greedy Algorithm – Basic Idea

- Always make **locally optimal** choice that looks best at the moment, hoping that this choice will lead to a **globally optimal** solution
- Does not always work, but sometimes it does
  - When it does, you have a very efficient solution that can be coded quickly

Coin Change (General Case)

- Given many coins of different denominations = {4, 3, 1} cents, make X cents with the *least number of coins* used!
  - If X = 6, greedy solution: 4+1+1 = 6 (3 coins)
  - But the optimal solution is: 3+3 = 6 (2 coins)
- This problem will be revisited in "Dynamic Programming" topic

Few Classic Greedy Problems

- Coin Change *certain denominations only*
- Fractional Knapsack
- Load Balancing
- Greedy Assignment
- Interval Covering/Activity Selection
- Huffman's Coding
- Minimum Spanning Tree
- +ve Weighted Single-Source Shortest Paths

Designing Greedy Algorithm (5)

- Tips (Not rule of thumb!)
  - In ICPC, if the input size is too large* for Complete Search or even DP, Greedy may be the (only) way
    - But if the input size is small, avoid greedy!
      Try complete search or DP first!
      - If our Complete Search or DP always select one particular (best) sub problem at each step, greedy strategy may works
  - If no obvious greedy strategy seen, try to sort the data first or introduce some tweaks
    - See if greedy strategy emerges

Greedy in Other Algorithms

- Greedy strategy is embedded in more complex algorithms:
  - +ve Weighted Single-Source Shortest Path
    - Dijkstra's
  - Minimum Spanning Tree
    - Kruskal's
    - Prim's
  - Combined with binary search the answer
    - See the next section
  - Etc

Interesting Usages of Binary Search

DIVIDE AND CONQUER

Divide and Conquer Paradigm

- Idea:
  - Divide: Split original problem into sub-problems
  - Conquer: Solve sub-problems and then use some (or all) the results of sub-problems to solve the original problem
- Data Structures/Algorithms with D&C flavor:
  - Merge/Quick/Heap Sort, findKthIndex, Binary Search, BST/Heap/Segment/Fenwick Tree, etc
- See if your problem can be solved using D&C!

Focus on Binary Search the Answer

- Ordinary Usage
  - On Static Sorted Array
    - C++ STL algorithm::lower_bound (Java Collections.binarySearch)
- Not so common (read textbook)
  - On weighted tree with root-to-leaf path sorted!
- Bisection (Numerical) Method (read textbook)
  - Find root x of a function s.t. f(x) = 0
    - Let computer do the work
- Binary Search the Answer (discussed next)

Binary Search the Answer (1)

- UVa 714 – Copying Books
  - Given m books with different pages p_i $\{p_1, p_2, \ldots, p_m\}$
  - Make a copy of each book with k scribes
  - Each book can only be assigned to a single scriber
  - Every scriber must get a continuous sequence of books, i.e. $0 = b_0 < b_1 < b_2 < \ldots < b_{k-1} < b_k = m$
  - Minimize the maximum number of pages assigned to a scriber
  - $1 \leq k, m \leq 500$, $1 \leq p_i \leq 10000000$
- Example:
  - m = 9, k = 3, p = {100,200,300,400,500,600,700,800,900}
  - Answer: 1700 pages: 100,200,300,400,500|600,700|800,900

Binary Search the Answer (2)

- Can we "guess" this answer in binary search fashion?
  - Yes
  - If we guess answer = 1000 pages, we have no solution
    - Greedy assignment!
      - Give as many consecutive books to current scribe until "overflow"
    - 100,200,300,400|500|600|**{700,800,900} (unassigned)**
    - Answer is too low, we have to increase the workload per each scriber
  - If we guess answer = 2000 pages, we have a solution + slack
    - 100,200,300,400,500|600,700|800,900 **(slack 500|700|300)**
    - Answer is too high, we can decrease the workload per each scriber
  - And so on until we hit answer = 1700 pages (optimal)

Summary

- We have seen:
  - Complete Search: Iterative and Recursive Backtracking
  - Greedy
  - Divide and Conquer: Focus on Binary Search the Answer
- Next week, we will see (revisit) the fourth paradigm:
  - Dynamic Programming

References

- **Competitive Programming**, chapter 3
  - Steven, Felix ☺
- **Introduction to Algorithms**, p370-404
  - Cormen, Leiserson, Rivest, Stein
- **Algorithm Design**, p115-208
  - Tardos, Kleinberg
- **Algorithms**, p127-155
  - Dasgupta, Papadimitirou, Vazirani