CS3233 Competitive Programming

Dr. Steven Halim

Week 03 – Problem Solving Paradigms

(Focus on Complete Search)

Outline

- Mini Contest #2 + Break + Discussion + Admins
- Complete Search
  - **Iterative: (Nested) Loops, Permutations, Subsets**
  - **Recursive Backtracking (N Queens), from easy to (very) hard**
  - **State-Space Search**
  - **Meet in the Middle (Bidirectional Search)**
- Read at home (will not be tested in mini contest A/B):
  - Some tips to speed up your solution
  - Greedy algorithms
  - Divide and Conquer (D&C) algorithms
  - Especially Binary Search the Answer technique

Iterative: (Nested) Loops, Permutations, Subsets

Recursive Backtracking (N Queens), from easy to (very) hard

State-Space Search

Meet in the Middle (Bidirectional Search)

COMPLETE SEARCH

Iterative Complete Search
Loops (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
Loops (2)

- More challenging variants:
  - 2-3-4-…-K nested loops
  - Some pruning are possible,
    e.g. using “continue”, “break”, or if-statements

Iterative Complete Search
Permutations

- UVa 11742 – Social Constraints
  - There are $0 < \mathbf{n} \le 8$ movie goers
  - They will sit in the front row with **n** consecutive open seats
  - There are $0 \le \mathbf{m} \le 20$ 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

Subsets

- 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

Iterative: (Nested) Loops, Permutations, Subsets

Recursive Backtracking (N Queens), from easy to (very) hard

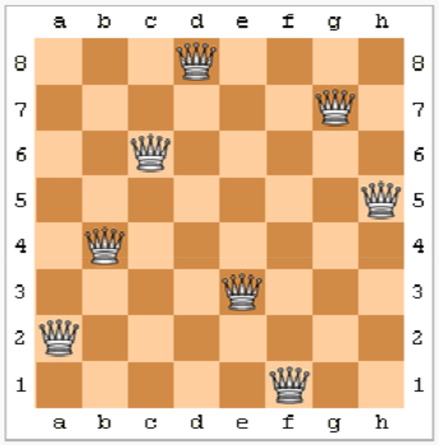
State-Space Search

Meet in the Middle (Bidirectional Search)

COMPLETE SEARCH

Recursive Backtracking (1)

- UVa 750 – 8 Queens Chess Problem
  - Put 8 queens in 8x8 Chessboard
  - No queen can attack other queens
- Naïve ways (Time Limit Exceeded)
  - Choose 8 out of 64 cells…
    - ${}_{64}C_8$ = 4 Billion possibilities… ☹
  - Insight 1: Put one queen in each column…
    - 8^8 = 17 Million possibilities… :O



One solution.

Recursive Backtracking (2)

- Better way, recursive backtracking
  - Insight 2: **all-different constraint** for the rows too
    - We put one queen in each column **AND each row**
    - Finding a valid permutation out of 8! possible permutations…
    - Search space goes down from 8^8 = 17M to 8! = 40K!
  - Insight 3: <u>**main**</u> diagonal <u>**and secondary diagonal**</u> 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!

Important Code (3)

```
int rw[8], TC, a, b, lineCounter;                  // ok to use global variables

bool place(int r, int c) {
  for (int prev = 0; prev < c; prev++)    // check previously placed queens
    if (rw[prev] == r || (abs(rw[prev] - r) == abs(prev - c)))
      return false;          // share same row or same diagonal -> infeasible
  return true; }

void backtrack(int c) {
  if (c == 8 && rw[b] == a) {            // candidate sol, (a, b) has 1 queen
    printf("%2d      %d", ++lineCounter, rw[0] + 1);
    for (int j = 1; j < 8; j++) printf(" %d", rw[j] + 1);
    printf("\n"); }
  for (int r = 0; r < 8; r++)                            // try all possible row
    if (place(r, c)) {          // if can place a queen at this col and row
      rw[c] = r; backtrack(c + 1);       // put this queen here and recurse
}   }
```

Is that the best n-Queens solution?

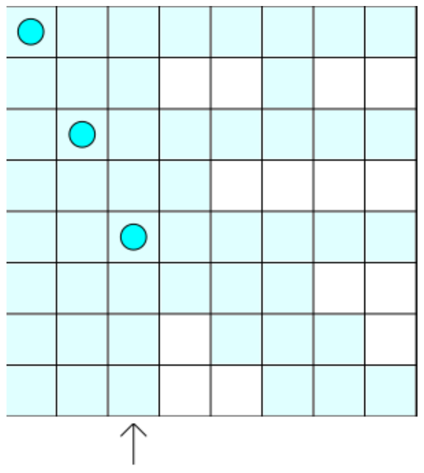
- Maybe not
  - See UVa 11195 – Another n-Queen Problem
- Several cells are forbidden
  - Do this helps?
- n can now be as large as n=14 :O??
  - How to run 14! algorithm in a few seconds?

Speeding Up Diagonal Checks

- This check is slow:

```
bool place(int r, int c) {
  for (int prev = 0; prev < c; prev++)    // check previously placed queens
    if (rw[prev] == r || (abs(rw[prev] - r) == abs(prev - c)))
      return false;        // share same row or same diagonal -> infeasible
  return true; }
```

- We can speed up this part by using 2*n-1 boolean arrays (or bitset) to test if a certain left/right diagonal can be used



http://www.comp.nus.edu.sg/~stevenha/visualization/recursion.html

Is that enough?

- Unfortunately no
- But fortunately there is a better way of using diagonal checks ☺

Iterative: (Nested) Loops, Permutations, Subsets

Recursive Backtracking (N Queens), from easy to (very) hard

State-Space Search

Meet in the Middle (Bidirectional Search)

COMPLETE SEARCH

UVa 11212 – Editing a Book

Rujia Liu's Problem

- Given *n* equal-length paragraphs numbered from 1 to *n*
- Arrange them in the order of 1, 2, ..., *n*
- With the help of a clipboard,
  you can press Ctrl-X (cut) and Ctrl-V (paste) several times
  - You cannot cut twice before pasting, but you can cut several contiguous paragraphs at the same time - they'll be pasted in order
- The question: What is the minimum number of steps required?
- Example 1: In order to make {2, 4, (1), 5, 3, 6} sorted,
  you can cut 1 and paste it before 2 → {**1,** 2, 4, 5, (3), 6}
  then cut 3 and paste it before 4 → {1, 2, **3,** 4, 5, 6} → done √
- Example 2: In order to make {(3, 4, 5), 1, 2} sorted,
  you can cut {3, 4, 5} and paste it after {1, 2} → {1, 2, **3, 4, 5**} √
  or cut {1, 2} and paste it before {3, 4, 5} → {**1, 2**, 3, 4, 5} √

Loose Upper Bound

- Answer: *k*-1
  - Where *k* is the number of paragraphs initially the wrong positions
- Trivial but wrong algorithm:
  - Cut a paragraph that is in the wrong position
  - Paste that paragraph in the correct position
  - After *k*-1 such cut-paste, we will have a sorted paragraph
    - The last wrong position will be in the correct position at this stage
  - But this may not be the shortest way
- Examples:
  - {(3), 2, 1} → {(2), 1, **3**} → {1, **2**, 3} → 2 steps
  - {(5), 4, 3, 2, 1} → {(4), 3, 2, 1, **5**} → {(3), 2, 1, **4**, 5} → {(2), 1, **3**, 4, 5} → {1, **2**, 3, 4, 5} → 4 steps

The Actual Answers

- {3, 2, 1}
  - Answer: 2 steps, e.g.
    - {(3), 2, 1} → {(2), 1, **3**} → {1, **2**, 3}, or
    - {3, 2, (1)} → {**1**, (3), 2,} → {1, 2, **3**}
- {5, 4, 3, 2, 1}
  - Answer: Only **3** steps, e.g.
    - {5, 4, (3, 2), 1} → {**3, (2**, 5), 4, 1} → {3, 4, (1, **2), 5**} → {**1, 2**, 3, 4, 5}
- How about {5, 4, 9, 8, 7, 3, 2, 1, 6}?
  - Answer: 4, but very hard to compute manually
- How about {9, 8, 7, 6, 5, 4, 3, 2, 1}?
  - Answer: 5, but very hard to compute manually

Some Analysis

- There are at most n! permutations of paragraphs
  - With maximum n = 9, this is 9! or 362880
  - The number of vertices is not that big actually
- Given a permutation of length n (a vertex)
  - There are ${}_nC_2$ possible cutting points (index i, j $\in$ [1..n])
  - There are n possible pasting points (index k $\in$ [1..(n-(j-i+1))])
  - Therefore, for each vertex, there are about $O(n^3)$ branches
- The worst case behavior if we run a single BFS on this State-Space graph is: $O(V+E) = O(n! + n!*n^3) = O(n!*n^3)$
  - With n = 9, this is $9! * 9^3$ = 264539520 ~ 265 M, TLE (or maybe MLE…)

Iterative: (Nested) Loops, Permutations, Subsets

Recursive Backtracking (N Queens), from easy to (very) hard

State-Space Search

Meet in the Middle (Bidirectional Search)

COMPLETE SEARCH

More Search Algorithms…

- Depth Limited Search (DLS) + Iterative DLS
- A* / Iterative Deepening A* (IDA*) / Memory Bounded A*
- Branch and Bound (BnB)
- Maybe in Week12 ☺ or ...
  - We will not test any of these in mini contests problem A/B

Summary

- We have seen some Complete Search techniques…
  - There are (several) others…
  - We still need lots of practice though ☺
- We "skipped" Greedy and Divide & Conquer this time
  - Read Section 3.3-3.4 by yourself
  - We will not test these on mini contests problem A/B
- Next week, we will see (revisit) the fourth paradigm:
  - Dynamic Programming

References

- **Competitive Programming 2.9**, Section 3.2 + 8.2
  - Steven, Felix ☺