CS 491 CAP
Intro to Competitive Algorithmic Programming

Lecture 4
Ad Hoc and Simulation

Uttam Thakore

University of Illinois at Urbana-Champaign

Sep 22, 2017

Tryouts

- Our first tryouts will be on Saturday, Sept 23

What are Ad Hoc and Simulation?

- Simulation: Do exactly what the problem statement tells you.
  - E.g. simulate a board game, such as UNO, Blockus, and King of Tokyo, after the statement tells you the rules and strategies (may be simplified).
- Ad Hoc (a.k.a. Bruteforce): The algorithm is very straightforward.
  - E.g. just enumerate all possible solutions and figure out the answer.
- Backtracking is also very useful in bruteforce search problem.
  - E.g. Exact Cover Problems, such as 8-Queens and Sudoku.

Josephus problem: Statement

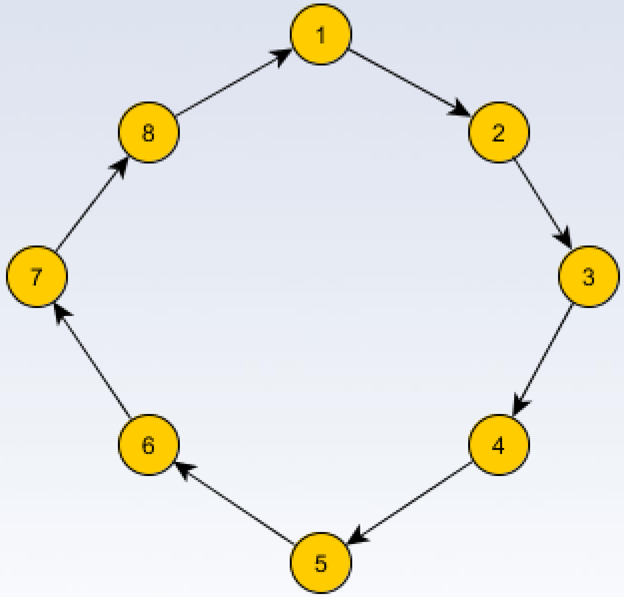
- N people standing in a circle, numbered from 1 to N
- The next person of i is $i \% N + 1$
  - E.g. the next person of 1 is 2
  - E.g. the next person of N is 1
- The counting out begins at s-th person
- In each step, $K - 1$ people are skipped and the next person is executed.
  - If we start with 1, the first one to be executed is K.
- Given N, K, and s, who is the last person?

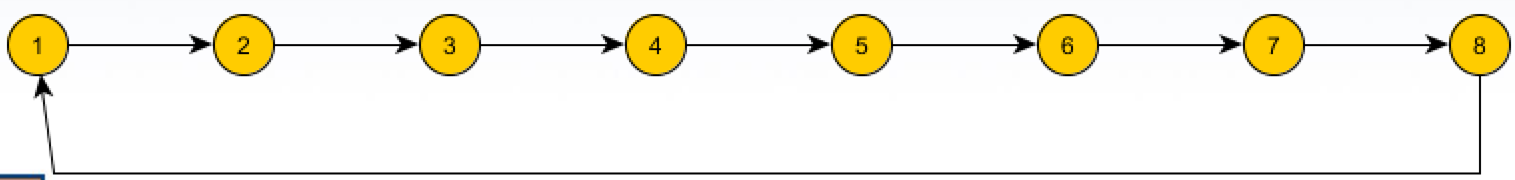
Josephus problem: Example

- Example, $N = 7, K = 2, s = 1$
- 0^{th} round: **1** 2 3 4 5 6 7
- 1^{st} round: 1 **3** 4 5 6 7
- 2^{nd} round: 1 3 **5** 6 7
- 3^{rd} round: 1 3 5 **7**
- 4^{th} round: **3** 5 7
- 5^{th} round: 3 **7**
- 6^{th} round: **7**

Josephus problem: Solution

- Data Structure: Circular Singly Linked List





Josephus problem: Solution

- Data Structure: Circular Singly Linked List
- Two for-loops to simulate the execution process

```
current_person = S-th person
for (int iteration = 1; iteration < N; ++
iteration) {
    for (int i = 1; i < K; ++ i) {
        current_person = move to next person
    }
    kill current_person
}
```

Josephus problem: Solution

- Data Structure: Circular Singly Linked List
- Two for-loops to simulate the execution process
- Time Complexity: $O(NK)$

Josephus problem: Practice

- POJ 1012
- http://poj.org/problem?id=1012

- Tips: There might be some duplicated test cases in a single run. You can store the answers in your program and print them when you see the same test cases again.

Josephus problem: Future

- You will learn more efficient solutions in Dynamic Programming, which are $O(N)$ and $O(KlogN)$.

Social Constraints: Statement

- There are $n \leq 8$ movie goers
- They will sit in the front row with n consecutive open seats
- There are $m \leq 20$ 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?

Social Constraints: Example

- 3 people in total.
- 1 constraint: 1 and 2 must sit adjacently.
- All possible assignments:
  - 1 2 3
  - 2 1 3
  - 3 1 2
  - 3 2 1

Social Constraints: Solution

- Try all possible seats assignment

```
vector<int> perm;
for (int i = 0; i < n; ++ i) {
   perm.push_back(i);
}
do {
   // check the conditions
} while (next_permutation(perm.begin(),
perm.end());
```

Social Constraints: Solution

- Try all possible seats assignment
- Time Complexity: $O(n!\,m)$
- Worse case: $8! \times 20 = 806{,}400$

Tips

- Usually, within 1 second running time, you can apply 10^7 multiplication operations. Sometimes, on some efficient machine, 10^8 is also fine.
- **`/`**, **`sqrt`**, **`cos`**, **`sin`**, **`atan2`**, and so on. These operations are a little slower.
- **`|`**, **`&`**, **`^`**, **`~`** are much faster.

Subset Enumeration

- How many subsets of {1..n} are “good”?
- “good” is very efficient and easy to judge after you have the subset.
- $n \leq 20$

Backtracking

- Save the states before recursion
- Restore the states after recursion

Subset Enumeration

- Backtracking:

```
vector<int> subset;
int n;
void search(int i)
{
    if (i == n) {
        // do something for subset ...
        return;
    }
    // choose i
    subset.push_back(i);
    dfs(i + 1);
    subset.pop_back(i);

    // not choose i
    dfs(i + 1);
}
```

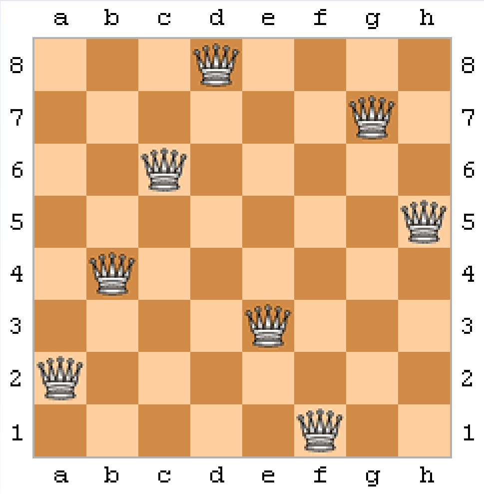
Subset Enumeration

- Let's use bitmask instead of backtracking.

```
for (int mask = 0; mask < 1 << n; ++ mask) {
    vector<int> subset;
    for (int i = 0; i < n; ++ i) {
        if (mask >> i & 1) {
            subset.push_back(i);
        }
    }
    // do something for subset ...
}
```

8 Queens (UVA 750)

- Print ALL possible solution for 8 Queens



8 Queens

- i-th step
  - For all 1 <= x <= 8
    - Try to put a new queen on (i, x)
    - Update the "attackness" of each grid
    - Recursion to (i+1)-th step
    - Remove (i, x)
    - Re-update the "attackness" of each grid

Sudoku (POJ 3074)

◇ Given a partially filled 9 x 9 Sudoku, find a possible solution.

◇ Every row/column/square contains 1-9 exactly once.

.	2	7	3	8	.	.	1	.
.	1	.	.	.	6	7	3	5
.	.	.	.	.	.	.	2	9
3	.	5	6	9	2	.	8	.
.	.	.	.	.	.	.	.	.
.	6	.	1	7	4	5	.	3
6	4	.	.	.	.	.	.	.
9	5	1	8	.	.	.	7	.
.	8	.	.	6	5	3	4	.

Sudoku

- (i, j)-th step
  - For all $1 <= d <= 9$
    - Try to put d on (i, j)
    - Update the "conflict table" of each grid
    - Recursion to (i', j')-th step
    - Reset (i, j) to unknown
    - Re-update the "conflict table" of each grid

DuLL: ICPC Mid-Central 2009

- http://www.icpc-midcentral.us/archives/2009/mcpc2009/dull/dull.html

DuLL: ICPC Mid-Central 2009

- Discrete Event Simulation
- Use *frequency dictionary* to keep track of how many programs rely on a DLL
- Iterate through *event list* (each program entry and exit) instead of over each time step
- At each event:
  - Update the frequency dictionary
  - Check if DLL memory usage is greater than previously seen

Summary

- In the following situations, it will be likely a simulation or ad hoc problem:
  - The statement tells what you need to do
    - E.g. tells you the detailed rules of a poker game
  - Ask for almost all (intermediate) results/solutions
    - E.g. print all primes between 1 to n
  - The size of data is quite small
    - Permutation: $O(n!)$ algorithm for $n \sim 10$
    - Subsets: $O(2^n)$ algorithm for $n \sim 20$
  - Backtracking when they ask for only ONE possible solutions
    - Exponential Time Complexity!

Questions?