Advanced Computer Contest Preparation
Lecture 18

# STATE COMPRESSION BITMASK DP

# Sample Problem: Travelling Salesman Problem

- There are $N$ cities connected by $M$ weighted roads
- You start in any city and end in any city
- You want to visit every city
- You can visit cities more than once
- What is the length of the shortest possible path?

# Solutions?

- Preprocess:
  - Run an all-pairs shortest path algorithm so we can get distance between any 2 cities in $O(1)$ time in the future
  - Floyd-Warshall runs in $O(N^3)$ time; very short code
- Brute force?
  - Try every possible permutation of cities to visit
  - Runtime: $O(N!)$

# State Compression

- DP revolves around the concept of subproblems, or states
- These states must be able to be represented in memory
  - Easy to convert to some array/map location
- Sometimes, states are complex
  - Many variables
  - Abstract variables
- State compression involves turning a complex state into a simpler representation

# Bitmask

- Used when the state has several variables and each can either be true (1) or false (0)
- A single integer is used to represent all of these variables
  - Integers are represented in binary in a computer
  - Each digit is called a bit
- Size of data type in bits corresponds to how many variables we can store
  - `short` = 16, `int` = 32, `long long` = 64
- Similar functionality to a `bool` array, but can be indexed by arrays and maps

# Bitwise Operations

- Bitwise AND (&)
  - Takes each corresponding bit and performs the AND function
  - e.g. 7 & 13 = 5
- Bitwise OR (|)
  - Takes each corresponding bit and performs the OR function
  - e.g. 5 | 9 = 13
- Bitwise XOR (^)
  - Takes each corresponding bit and performs the XOR function
  - e.g. 7 ^ 13 = 10

# Bitwise Operations

- Complement (~)
  - Flips all bits
- Left shift (<<)
  - Takes bits and shifts it left a specified number of times
  - Same as doubling repeatedly
  - e.g. 11 << 2 = 44
- Right shift (>>)
  - Takes bits and shifts it right a specified number of times
  - Same as halving repeatedly
  - e.g. 45 >> 3 = 5

# Bitmask Tips

- Check if bit is set (equal to 1)
  - `if (n & (1 << k))`
- Set a bit
  - `n |= (1 << k)`
- Unset a bit
  - `n &= ~(1 << k)`
- Flip a bit
  - `n ^= (1 << k)`
- Get number with first k bits set
  - `(1 << k) - 1`

# Travelling Salesman – DP Solution

- The overall problem is:
  - What is the length of the shortest path that visits all cities?
- The subproblems are:
  - If the last city we visited was city $i$, what is the length of the shortest path that visits some subset of the $N$ cities?
  - We will use a bitmask to represent which cities have been visited
    - If bit $i$ is set, city $i$ has been visited

# DP Solution

- Let $p(i,st)$ be the minimum distance of a path ending on city $i$ that visits all nodes with bits set in $st$
- Copy $st$ into another variable, called $st2$, and unset bit $i$
- Base case: $p(i,st) = 0$ if no bits are set in $st2$ $(st2 = 0)$
  - Explanation: city $i$ is the first city visited
- $p(i,st) = min(p(j,st2)+dist(j,i))$ if bit $j$ is set in $st2$
  - Explanation: go to city $i$ from city $j$ with state $st2$

# Pseudocode – Recursive

```
int solve(int node, int state){
        if (dp[node][state] != INF) return dp[node][state];
        int state2 = state & (~(1 << node));
        if (!state2) return dp[node][state] = 0;
        for (int i = 0; i < N; i++){
                if (state2 & (1 << i)){
                        dp[node][state] = min(dp[node][state],
                                solve(i,state2) + dist[i][node]);
                }
        }
        return dp[node][state];
}
//get input, init DP array to INF, do all pairs shortest path (Floyd-Warshall)
int ans = INF;
for (int i = 0; i < N; i++) ans = min(ans,solve(i,(1 << N)-1);
print(ans);
```

# Analysis

- How many states are there?
  - $O(N \times 2^N)$
- How many subproblems does each state depend on?
  - $O(N)$
- What is the time complexity needed to compute the solution to a problem?
  - $O(N)$
- Therefore, the final time complexity is $O(N^2 \times 2^N)$

# THANK YOU!