Game Theory and Bitmasks

Game Theory

Today we will be looking at games that:

- Are deterministic (no random factors)
- Are played by 2 players
- Both players play optimally
- Always end after a finite number of turns

Sample Problem

There is a set $A = \{a_1, a_2, ..., a_N\}$ consisting of N positive integers. Taro and Jiro will play the following game against each other.

Initially, we have a pile consisting of K stones. The two players perform the following operation alternately, starting from Taro:

- Choose an element x in A, and remove exactly x stones from the pile.

A player loses when he becomes unable to play. Assuming that both players play optimally, determine the winner.

Constraints

- All values in input are integers.
- $1 \leq N \leq 100$
- $1 \leq K \leq 10^5$
- $1 \leq a_1 < a_2 < \cdots < a_N \leq K$

https://dmoj.ca/problem/dpk

Solution

The number of stones remaining (and the person whose turn it is) uniquely defines the state of the game.

Each state is either winning or losing. We call a state “winning” if the person whose turn it is can win no matter how their opponent plays. We call a state “losing” if the person whose turn it is cannot win, assuming their opponent plays optimally.

For example, A = {1, 2, 5}, then the state where there are 2 stones left is a winning state because the person whose turn it is can remove both of them and win the game.

Whenever a player makes a move, the state of the game changes and it becomes the opponent’s turn.

Therefore, a state is winning if the current player can change it to a losing state and a state is losing if the current player can only change it to a winning state.

Solution

The number of stones remaining (and the person whose turn it is) uniquely defines the state of the game.

Each state is either winning or losing. We call a state “winning” if the person whose turn it is can win no matter how their opponent plays. We call a state “losing” if the person whose turn it is cannot win, assuming their opponent plays optimally.

For example, A = {1, 2, 5}, then the state where there are 2 stones left is a winning state because the person whose turn it is can remove both of them and win the game.

Whenever a player makes a move, the state of the game changes and it becomes the opponent’s turn.

Therefore, a state is winning if the current player can change it to a losing state and a state is losing if the current player can only change it to a winning state.

We store an array isWinning[] such that isWinning[i] is true iff a person can win if it’s their turn and there are i stones left.

isWinning[0] is false by definition.

We note that to calculate isWinning[i], we only need to know values for isWinning[j] where j < i. Therefore, we can iterate through all values of i (1 <= i <= K) from least to greatest and calculate isWinning[i].

Example

A = {1, 2, 5, 9}

10	L
9	W
8	W
7	W
6	L
5	W
4	W
3	L
2	W
1	W
0	L

Implementation

C++

```cpp
int n, k;
int arr[100];
bool isWinning[100001];

int main(){
    scanf("%d %d", &n, &k);
    for(int i = 0; i < n; i++) scanf("%d", arr + i);
    for(int i = 1; i <= k; i++){
        for(int j = 0; j < n; j++){
            if(i - arr[j] < 0) break;
            if(!isWinning[i-arr[j]]){
                isWinning[i] = true;
                break;
            }
        }
    }
    printf(isWinning[k] ? "First\n" : "Second\n");
}
```

Python

```python
n, k = map(int, input().split())
arr = list(map(int, input().split()))

isWinning = [False] * (k+1)

for i in range(k+1):
    for a in arr:
        if i - a < 0:
            break
        if not isWinning[i - a]:
            isWinning[i] = True
            break

print("First" if isWinning[k] else "Second")
```

Implementation

Java

```java
int n, k;
int[] arr;
boolean[] isWinning;

public Main() throws IOException {
    FastReader fr = new FastReader();
    n = fr.nextInt();
    k = fr.nextInt();
    arr = new int[n];
    isWinning = new boolean[k+1];

    for(int i = 0; i < n; i++) arr[i] = fr.nextInt();
    for(int i = 1; i <= k; i++){
        for(int j = 0; j < n; j++){
            if(i - arr[j] < 0) break;
            if(!isWinning[i-arr[j]]){
                isWinning[i] = true;
                break;
            }
        }
    }
    System.out.println(isWinning[k] ? "First" : "Second");
}

static class FastReader {...}
```

Recursive Approach

C++

```cpp
int n, k;
int arr[100];
// -1 if not calculated, 0 if losing, 1 if winning
int8_t isWinning[100001];

bool calcWinning(int state){
    if(isWinning[state] != -1) return isWinning[state];
    for (int i = 0; i < n; i++) {
        int a = arr[i];
        if(state - a < 0) break;
        if(!calcWinning(state - a)){
            isWinning[state] = 1;
            return true;
        }
    }
    isWinning[state] = 0;
    return false;
}

int main(){
    scanf("%d %d", &n, &k);
    for(int i = 0; i < n; i++) scanf("%d", arr + i);
    fill(isWinning, isWinning + k+1, -1);
    printf(calcWinning(k) ? "First\n" : "Second\n");
}
```

Python

```python
sys.setrecursionlimit(100005)

n, k = map(int, input().split())
arr = list(map(int, input().split()))
isWinning = [-1] * (k+1)

def calcWinning(state):
    if isWinning[state] != -1: return isWinning[state]
    for i in range(n):
        a = arr[i]
        if state - a < 0: break
        if not calcWinning(state - a):
            isWinning[state] = 1
            return True

    isWinning[state] = 0
    return False

print("First" if calcWinning(k) else "Second")
```

Recursive Approach

Java

```java
int n, k;
int[] arr;
byte[] isWinning;

boolean calcWinning(int state){
    if(isWinning[state] != -1) return isWinning[state] == 1;
    for (int i = 0; i < n; i++) {
        int a = arr[i];
        if(state - a < 0) break;
        if(!calcWinning(state - a)){
            isWinning[state] = 1;
            return true;
        }
    }
    isWinning[state] = 0;
    return false;
}

public Main() throws IOException {
    FastReader fr = new FastReader();
    n = fr.nextInt();
    k = fr.nextInt();
    arr = new int[n];
    isWinning = new byte[k+1];
    for(int i = 0; i < n; i++) arr[i] = fr.nextInt();
    System.out.println(calcWinning(k) ? "First" : "Second");
}

static class FastReader {...}
```

Recursive vs. Iterative

Iterative programs are generally faster. This is because they access memory in a more predictable way, don’t have overheads for calling functions (not very significant in C++, but function overheads are quite noticeable in Python), can sometimes take advantage of SIMD (single instruction multiple data) instructions, don’t use as much space on the stack (more relevant for memory usage), etc.

However, recursive solutions are sometimes better if you know that your program likely won’t be calculating every single state.

Key Takeaways

- A game can be represented as a set of finitely many states.
- Each state of a game will be either a winning or a losing state.
- A state is winning if at least one of the states it can be changed to is a losing state.

Another Problem

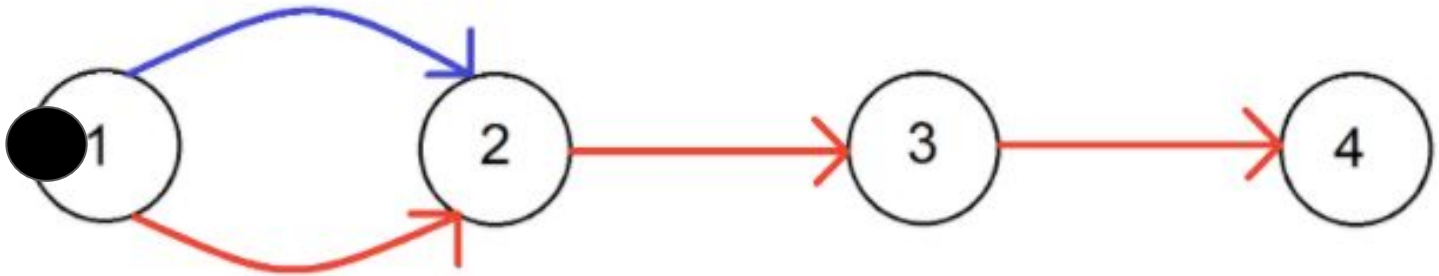
https://dmoj.ca/problem/qccp1

You are given a Directed Acyclic Graph (DAG) with N ($2 <= N <= 10^5$) nodes and M ($1 <= M <= 10^6$) edges. Each edge is one of 7 colors (r, o, y, g, b, i or v). There is a token on the graph, initially at node 1. There are two players, who alternate taking turns. In one turn, the player moves the token from the current node to another node by moving it along an edge. The game ends when the token has visited all the colours on the graph, or it is impossible to make a move. The winner is then the last player who moved the token. Determine who will win the game.

Another Problem

https://dmoj.ca/problem/qccp1

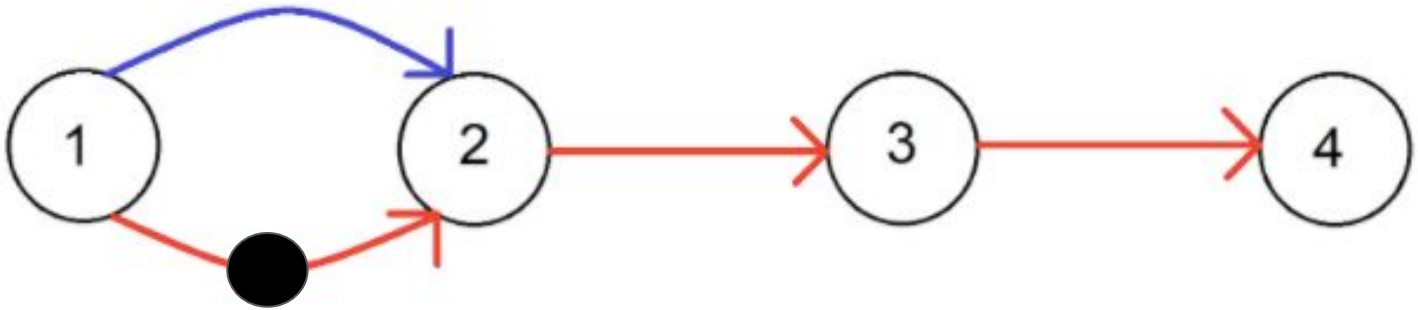
You are given a Directed Acyclic Graph (DAG) with N ($2 <= N <= 10^5$) nodes and M ($1 <= M <= 10^6$) edges. Each edge is one of 7 colors (r, o, y, g, b, i or v). There is a token on the graph, initially at node 1. There are two players, who alternate taking turns. In one turn, the player moves the token from the current node to another node by moving it along an edge. The game ends when the token has visited all the colours on the graph, or it is impossible to make a move. The winner is then the last player who moved the token. Determine who will win the game.



Another Problem

https://dmoj.ca/problem/qccp1

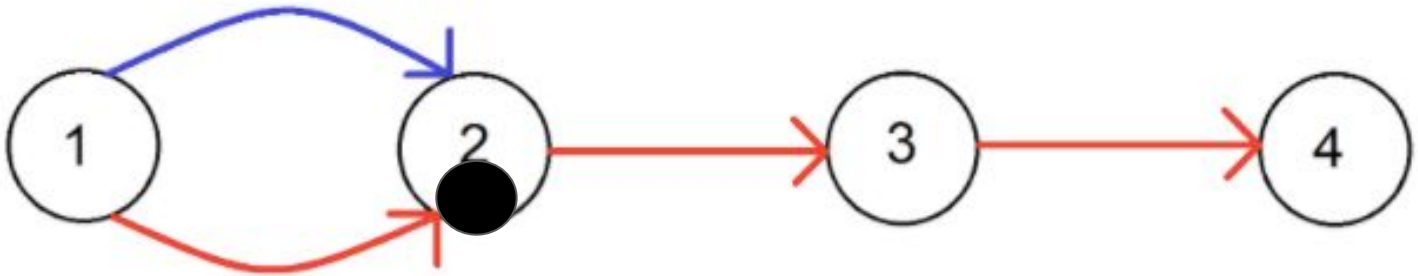
You are given a Directed Acyclic Graph (DAG) with N ($2 <= N <= 10^5$) nodes and M ($1 <= M <= 10^6$) edges. Each edge is one of 7 colors (r, o, y, g, b, i or v). There is a token on the graph, initially at node 1. There are two players, who alternate taking turns. In one turn, the player moves the token from the current node to another node by moving it along an edge. The game ends when the token has visited all the colours on the graph, or it is impossible to make a move. The winner is then the last player who moved the token. Determine who will win the game.



Another Problem

https://dmoj.ca/problem/qccp1

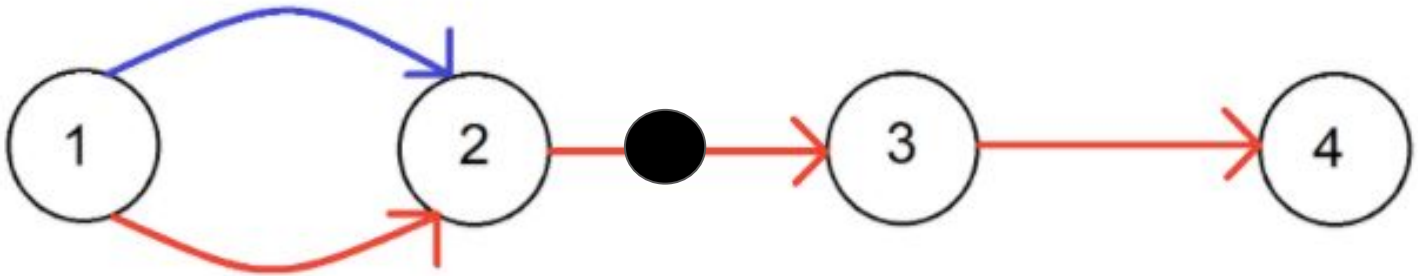
You are given a Directed Acyclic Graph (DAG) with N ($2 <= N <= 10^5$) nodes and M ($1 <= M <= 10^6$) edges. Each edge is one of 7 colors (r, o, y, g, b, i or v). There is a token on the graph, initially at node 1. There are two players, who alternate taking turns. In one turn, the player moves the token from the current node to another node by moving it along an edge. The game ends when the token has visited all the colours on the graph, or it is impossible to make a move. The winner is then the last player who moved the token. Determine who will win the game.



Another Problem

https://dmoj.ca/problem/qccp1

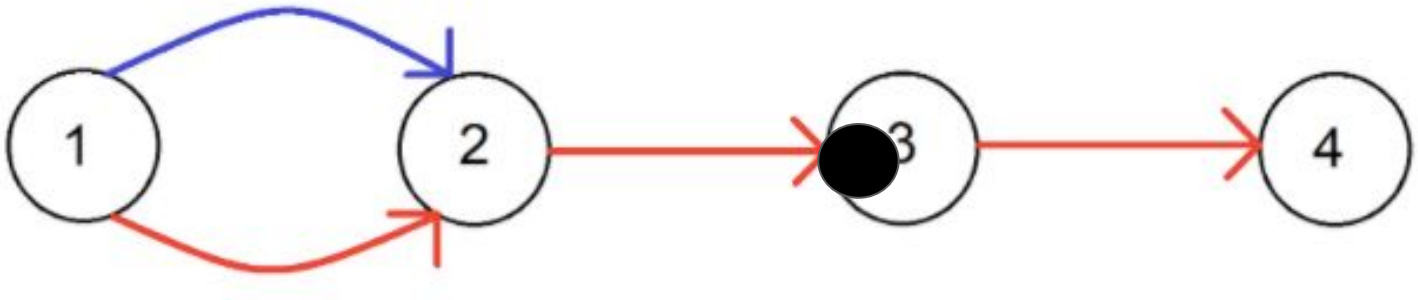
You are given a Directed Acyclic Graph (DAG) with N ($2 <= N <= 10^5$) nodes and M ($1 <= M <= 10^6$) edges. Each edge is one of 7 colors (r, o, y, g, b, i or v). There is a token on the graph, initially at node 1. There are two players, who alternate taking turns. In one turn, the player moves the token from the current node to another node by moving it along an edge. The game ends when the token has visited all the colours on the graph, or it is impossible to make a move. The winner is then the last player who moved the token. Determine who will win the game.



Another Problem

https://dmoj.ca/problem/qccp1

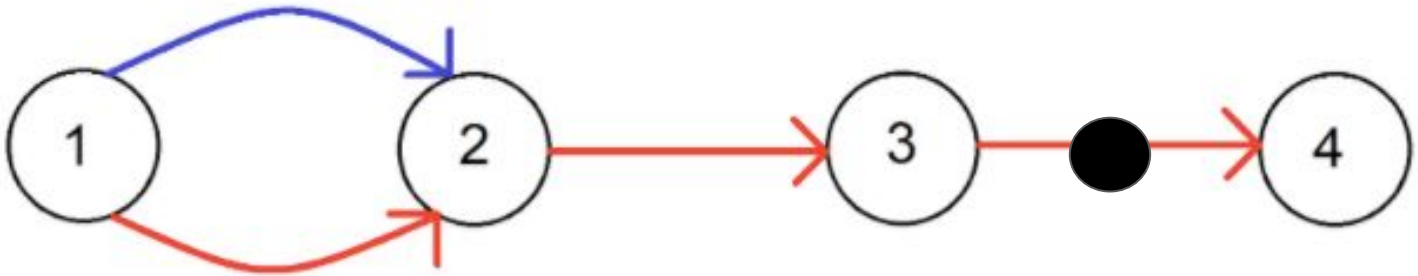
You are given a Directed Acyclic Graph (DAG) with N ($2 <= N <= 10^5$) nodes and M ($1 <= M <= 10^6$) edges. Each edge is one of 7 colors (r, o, y, g, b, i or v). There is a token on the graph, initially at node 1. There are two players, who alternate taking turns. In one turn, the player moves the token from the current node to another node by moving it along an edge. The game ends when the token has visited all the colours on the graph, or it is impossible to make a move. The winner is then the last player who moved the token. Determine who will win the game.



Another Problem

https://dmoj.ca/problem/qccp1

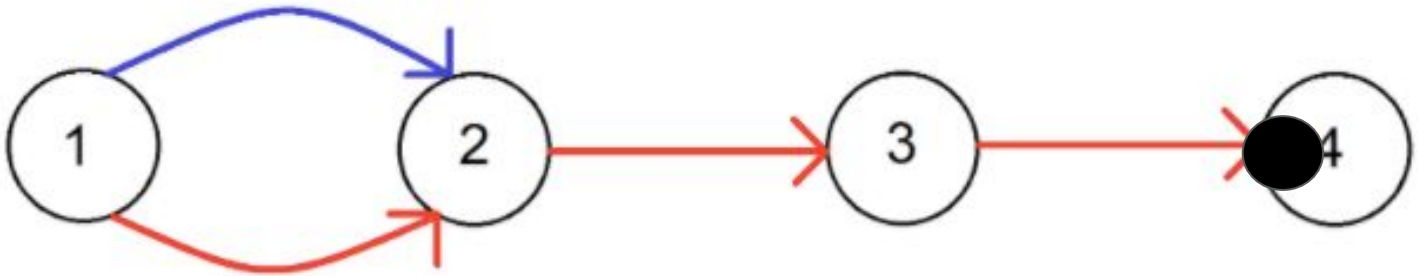
You are given a Directed Acyclic Graph (DAG) with N ($2 <= N <= 10^5$) nodes and M ($1 <= M <= 10^6$) edges. Each edge is one of 7 colors (r, o, y, g, b, i or v). There is a token on the graph, initially at node 1. There are two players, who alternate taking turns. In one turn, the player moves the token from the current node to another node by moving it along an edge. The game ends when the token has visited all the colours on the graph, or it is impossible to make a move. The winner is then the last player who moved the token. Determine who will win the game.



Another Problem

https://dmoj.ca/problem/qccp1

You are given a Directed Acyclic Graph (DAG) with N ($2 <= N <= 10^5$) nodes and M ($1 <= M <= 10^6$) edges. Each edge is one of 7 colors (r, o, y, g, b, i or v). There is a token on the graph, initially at node 1. There are two players, who alternate taking turns. In one turn, the player moves the token from the current node to another node by moving it along an edge. The game ends when the token has visited all the colours on the graph, or it is impossible to make a move. The winner is then the last player who moved the token. Determine who will win the game.

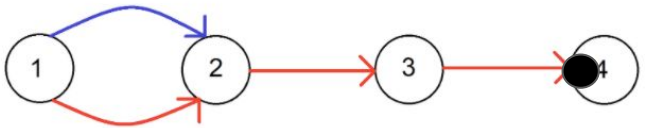


Another Problem

https://dmoj.ca/problem/qccp1

You are given a Directed Acyclic Graph (DAG) with N ($2 <= N <= 10^5$) nodes and M ($1 <= M <= 10^6$) edges. Each edge is one of 7 colors (r, o, y, g, b, i or v). There is a token on the graph, initially at node 1. There are two players, who alternate taking turns. In one turn, the player moves the token from the current node to another node by moving it along an edge. The game ends when the token has visited all the colours on the graph, or it is impossible to make a move. The winner is then the last player who moved the token. Determine who will win the game.

State: position of the token and the set of colors it has visited so far. There are 2^7 ways of choosing a subset of 7 colors. Therefore, there at most $N*2^7$ = 12,800,000 states in this game. This should easily fit inside the memory if we use for example 1 byte for each state.



Another Problem

https://dmoj.ca/problem/qccp1

You are given a Directed Acyclic Graph (DAG) with N ($2 <= N <= 10^5$) nodes and M ($1 <= M <= 10^6$) edges. Each edge is one of 7 colors (r, o, y, g, b, i or v). There is a token on the graph, initially at node 1. There are two players, who alternate taking turns. In one turn, the player moves the token from the current node to another node by moving it along an edge. The game ends when the token has visited all the colours on the graph, or it is impossible to make a move. The winner is then the last player who moved the token. Determine who will win the game.

State: position of the token and the set of colors it has visited so far. There are 2^7 ways of choosing a subset of 7 colors. Therefore, there at most $N*2^7$ = 12,800,000 states in this game. This should easily fit inside the memory if we use for example 1 byte for each state.

For each node, we will store an array of length 128, with each element corresponding to one of the states that have the token at that node. To determine what state each index corresponds to, we look at the binary representation of the index. If the i^{th} bit from the right is a 1, then the index corresponds to a state where the i^{th} token has already traveled across an edge with the i^{th} color.

Another Problem

https://dmoj.ca/problem/qccp1

You are given a Directed Acyclic Graph (DAG) with N ($2 <= N <= 10^5$) nodes and M ($1 <= M <= 10^6$) edges. Each edge is one of 7 colors (r, o, y, g, b, i or v). There is a token on the graph, initially at node 1. There are two players, who alternate taking turns. In one turn, the player moves the token from the current node to another node by moving it along an edge. The game ends when the token has visited all the colours on the graph, or it is impossible to make a move. The winner is then the last player who moved the token. Determine who will win the game.

State: position of the token and the set of colors it has visited so far. There are 2^7 ways of choosing a subset of 7 colors. Therefore, there at most $N*2^7$ = 12,800,000 states in this game. This should easily fit inside the memory if we use for example 1 byte for each state.

This will actually be a 2D array with size N by 128

For each node, we will store an array of length 128, with each element corresponding to one of the states that have the token at that node. To determine what state each index corresponds to, we look at the binary representation of the index. If the i^{th} bit from the right is a 1, then the index corresponds to a state where the i^{th} token has already traveled across an edge with the i^{th} color.

Yes, we give each color a number.

Another Problem

State: position of the token and the set of colors it has visited so far. There are 2^7 ways of choosing a subset of 7 colors. Therefore, there at most $N*2^7 = 12{,}800{,}000$ states in this game. This should easily fit inside the memory if we use for example 1 byte for each state.

For each node, we will store an array of length 128, with each element corresponding to one of the states that have the token at that node. To determine what state each index corresponds to, we look at the binary representation of the index. If the i^{th} bit from the right is a 1, then the index corresponds to a state where the i^{th} token has already traveled across an edge with the i^{th} color.

For example, 27 is 0011011 in binary (padded to 7 bits) which means that for each node, index 27 of our array will represent a state where colors 0, 1, 3, and 4 have already been visited but colors 2, 5, and 6 have not.

Another Problem

State: position of the token and the set of colors it has visited so far. There are 2^7 ways of choosing a subset of 7 colors. Therefore, there at most $N*2^7 = 12{,}800{,}000$ states in this game. This should easily fit inside the memory if we use for example 1 byte for each state.

For each node, we will store an array of length 128, with each element corresponding to one of the states that have the token at that node. To determine what state each index corresponds to, we look at the binary representation of the index. If the i^{th} bit from the right is a 1, then the index corresponds to a state where the i^{th} token has already traveled across an edge with the i^{th} color.

For example, 27 is 0011011 in binary (padded to 7 bits) which means that for each node, index 27 of our array will represent a state where colors 0, 1, 3, and 4 have already been visited but colors 2, 5, and 6 have not.

The base states in this game are nodes with no out-edges (losing; if it's your turn and the token is on a node with no out-edges then you lose) and states where all colors have already been visited (also losing).

Bitwise Operators

`a >> b` - bitwise right shift - shifts every bit in `a` to the right by `b` positions
`a << b` - bitwise left shift - shifts every bit in `a` to the left by `b` positions
`a & b` - bitwise AND - performs the logical AND operator on each bit of `a` and `b`
`a | b` - bitwise OR - performs the logical OR operator on each bit of `a` and `b`
`a ^ b` - bitwise XOR - performs the logical XOR operator on each bit of `a` and `b`
`~a` - bitwise not - flips all the bits in the number
Treating integers as booleans - in c++/python, any integer other than 0 is true and 0 is false. Java doesn't support casting between int and boolean.

Examples

0101 << 1 is 1010

0101 & 0110 is 0100

0110 ^ 1010 is 1100

Common operations

Set the i^{th} bit to 1:
`a |= 1 << i`

Retrieve the i^{th} bit:
`((a >> i) & 1)`

Create a number with the rightmost i bits set to 1:
`((1 << i) - 1)`

Note that the order of operations of bitwise operators is usually not what you want, so I would recommend using parentheses when writing complicated expressions.

Implementation

C++

```cpp
const int MN = 1e5;
vector<pair<int, int>> adjList[MN]; // {node, color}
int n, m;
int8_t dp[MN][128]; // -1 is uncalced, 0 is lose, 1 is win
int8_t FULL; // bitmask of a state with all colors

bool isWinning(int n1, int8_t s1){
    if(s1 == FULL || adjList[n1].empty()) return false;
    if(dp[n1][s1] != -1) return dp[n1][s1];
    for(auto [n2, c] : adjList[n1]){
        int8_t s2 = s1 | c;
        if(!isWinning(n2, s2)) return dp[n1][s1] = true;
    }
    return dp[n1][s1] = false;
}
```

```cpp
int main(){
    memset(dp, -1, sizeof(dp));
    cin.sync_with_stdio(0); cin.tie(0);
    cin >> n >> m;
    for (int i = 0; i < m; ++i) {
        int a, b;
        char c;
        cin >> a >> b >> c;
        a--; b--;
        if(c == 'r') c = 1;
        else if(c == 'o') c = 2;
        else if(c == 'y') c = 4;
        else if(c == 'g') c = 8;
        else if(c == 'b') c = 16;
        else if(c == 'i') c = 32;
        else if(c == 'v') c = 64;
        adjList[a].pb({b, c});
        FULL |= c;
    }

    cout << (isWinning(0, 0) ? "piddddgy\n" : "deruikong\n");
}
```

Implementation

Python

```
sys.setrecursionlimit(100005)

n, m = map(int, input().split())
adjList = [[] for i in range(n)]
dp = [[-1]*128 for i in range(n)]
FULL = 0

def isWinning(n1, s1):
    if s1 == FULL or not adjList[n1]: return False
    if dp[n1][s1] != -1: return dp[n1][s1]
    for n2, c in adjList[n1]:
        s2 = s1 | c
        if not isWinning(n2, s2):
            dp[n1][s1] = 1
            return True
    dp[n1][s1] = 0
    return False
```

```
for i in range(m):
    line = input().split()
    a = int(line[0]) - 1
    b = int(line[1]) - 1
    c = line[2]

    if c == 'r': c = 1
    elif c == 'o': c = 2
    elif c == 'y': c = 4
    elif c == 'g': c = 8
    elif c == 'b': c = 16
    elif c == 'i': c = 32
    elif c == 'v': c = 64

    adjList[a].append((b, c))
    FULL |= c

print("piddddgy" if isWinning(0, 0) else "deruikong")
```

Implementation

Java

```java
int n, m;
ArrayList<Edge>[] adjList;
byte[][] dp;
byte FULL;

boolean isWinning(int n1, byte s1){
    if(s1 == FULL || adjList[n1].isEmpty()) return false;
    if(dp[n1][s1] != -1) return dp[n1][s1] == 1;
    for(Edge e : adjList[n1]){
        byte s2 = (byte) (s1 | e.color);
        if(!isWinning(e.node, s2)){
            dp[n1][s1] = 1;
            return true;
        }
    }
    dp[n1][s1] = 0;
    return false;
}

static class Edge {
    int node;
    byte color;

    Edge(int node, byte color){
        this.node = node;
        this.color = color;
    }
}
```

```java
public Main() throws IOException {
    FastReader fr = new FastReader();
    n = fr.nextInt();
    m = fr.nextInt();

    dp = new byte[n][128];
    adjList = new ArrayList[n];
    for (int i = 0; i < n; i++) {
        Arrays.fill(dp[i], (byte) -1);
        adjList[i] = new ArrayList<>();
    }

    for (int i = 0; i < m; ++i) {
        int a = fr.nextInt();
        int b = fr.nextInt();
        byte c = fr.readByte(); fr.readByte();
        a--; b--;
        if(c == 'r') c = 1;
        else if(c == 'o') c = 2;
        else if(c == 'y') c = 4;
        else if(c == 'g') c = 8;
        else if(c == 'b') c = 16;
        else if(c == 'i') c = 32;
        else if(c == 'v') c = 64;
        adjList[a].add(new Edge(b, c));
        FULL |= c;
    }

    System.out.println(isWinning(0, (byte) 0) ? "piddddgy" : "deruikong");
}

static class FastReader {...}
```

Bitmasks for Non-Game Theory Problems

More Bitmasks

Encoding information in the binary representation on an integer can be helpful for other, non game theory, problems as well.

Longest Path Problem

There are many well-known algorithms for finding the shortest route from one location to another. People have GPS devices in their cars and in their phones to show them the fastest way to get where they want to go. While on vacation, however, Troy likes to travel slowly. He would like to take the *longest* route to his destination so that he can visit many new and interesting places along the way.

As such, a valid route consists of a sequence of distinct cities, $c_1, c_2, ..., c_k$, such that there is a road from c_i to c_{i+1} for each $1 \leq i < k$.

He does not want to visit any city more than once. Can you help him find the longest route?

Input Specification

The first line of input contains two integers n, m, the total number of cities and the number of roads connecting the cities ($2 \leq n \leq 18$; $1 \leq m \leq n^2 - n$). There is at most one road from any given city to any other given city. Cities are numbered from 0 to $n - 1$, with 0 being Troy's starting city, and $n - 1$ being his destination.

The next m lines of input each contain three integers s, d, l. Each triple indicates that there is a one way road from city s to city d of length l km ($0 \leq s \leq n - 1$; $0 \leq d \leq n - 1$; $s \neq d$; $1 \leq l \leq 10\,000$). Each road is one-way: it can only be taken from s to d, not vice versa. There is always at least one route from city 0 to city $n - 1$.

For at least 30% of the marks for this problem, $n \leq 8$.

Output Specification

Output a single integer, the length of the longest route that starts in city 0, ends in city $n - 1$, and does not visit any city more than once. The length is the sum of the lengths of the roads taken along the route.

https://dmoj.ca/problem/cco15p2

Longest Path Problem

- Find the longest path in a graph that starts at node 0 and ends at node N-1
- You can visit each node at most once
- N is at most 18

18! is around $6 * 10^{15}$, so brute forcing all possible paths would probably TLE.

Bitmask Dynamic Programming Approach

- Find the longest path in a graph that starts at node 0 and ends at node N-1
- You can visit each node at most once
- N is at most 18

Create an array dp[n][2^n] where dp[i][j] is the maximum distance you can travel assuming you end at node i and you have visited all other nodes that have a set bit in j.

For example, consider dp[3][19]. 19 = 10011_2. Therefore, dp[3][19] will represent the maximum distance you can travel assuming you ended at node 3 and visited only nodes 0, 1, and 4 previously.

Consider all the transitions we can make from dp[i][j].
We can go to node k only if the k^{th} bit of j isn't set, and there is an edge from i -> k, whose weight we denote with w.
Set `dp[k][j | (1 << i)] = max(dp[k][j | 1 << i], dp[i][j] + w);`

We calculate the dp states in increasing order of j.
DP states can also be calculated in the other direction and/or recursively, which in hindsight, might have been easier to explain.

Implementation

C++

```cpp
const int MN = 18;
int n, m;
int dp[MN][1 << MN];
vector<pair<int, int>> adjList[MN]; // {node, weight}

int main(){
    cin.sync_with_stdio(0); cin.tie(0);
    cin >> n >> m;
    for (int i = 0; i < m; ++i) {
        int a, b, c;
        cin >> a >> b >> c;
        adjList[a].push_back({b, c});
    }

    int maxState = 1 << n;
    memset(dp, -1, sizeof(dp));
    dp[0][0] = 0;

for (int state = 0; state < maxState; ++state) {
        for (int n1 = 0; n1 < n; ++n1) {
            int cur = dp[n1][state];
            if(cur == -1 || ((state >> n1) & 1)) continue;
            int newState = state | (1 << n1);

            for(auto[n2, w] : adjList[n1]){
                // If n2 was already visited, skip
                if((newState >> n2) & 1) continue;
                dp[n2][newState] = max(dp[n2][newState], cur + w);
            }
        }
    }

    int ans = 0;
    for (int i = 0; i < maxState; ++i) ans = max(ans, dp[n-1][i]);
    cout << ans << '\n';
}
```

Implementation

Python

```python
n, m = map(int, input().split())
adjList = [[] for i in range(n)]

for i in range(m):
    a, b, c = map(int, input().split())
    adjList[a].append((b, c))

maxState = 1 << n
dp = [[-1 for i in range(maxState)] for j in range(n)]
dp[0][0] = 0

for state in range(maxState):
    for n1 in range(n):
        cur = dp[n1][state]
        if cur == -1 or ((state >> n1) & 1): continue
        newState = state | (1 << n1)

        for n2, w in adjList[n1]:
            if (newState >> n2) & 1: continue
            dp[n2][newState] = max(dp[n2][newState], cur + w)

ans = max(dp[n-1][i] for i in range(maxState))
print(ans)
```

Implementation

Java

```java
int n, m;
int[][] dp;
ArrayList<Edge>[] adjList; // {node, weight}

public Main() throws IOException {
    FastReader fr = new FastReader();
    n = fr.nextInt();
    m = fr.nextInt();
    adjList = new ArrayList[n];
    for (int i = 0; i < n; i++) adjList[i] = new ArrayList<>();
    for (int i = 0; i < m; ++i) {
        int a = fr.nextInt();
        int b = fr.nextInt();
        int c = fr.nextInt();
        adjList[a].add(new Edge(b, c));
    }

    int maxState = 1 << n;
    dp = new int[n][maxState];
    for (int i = 0; i < n; i++) Arrays.fill(dp[i], -1);
    dp[0][0] = 0;

    for (int state = 0; state < maxState; ++state) {
        for (int n1 = 0; n1 < n; ++n1) {
            int cur = dp[n1][state];
            if(cur == -1 || ((state >> n1) & 1) == 1) continue;
            int newState = state | (1 << n1);

            for(Edge e : adjList[n1]){
                if(((newState >> e.node) & 1) == 1) continue;
                dp[e.node][newState] = Math.max(dp[e.node][newState], cur
+ e.weight);
            }
        }
    }

    int ans = 0;
    for (int i = 0; i < maxState; ++i) ans = Math.max(ans, dp[n-1][i]);
    System.out.println(ans);
}

static class Edge {
    int node, weight;
    Edge(int node, int weight){
        this.node = node;;
        this.weight = weight;
    }
}

static class FastReader {...}
```

Exercises:

Problems discussed in these slides:
https://dmoj.ca/problem/dpk
https://dmoj.ca/problem/qcc1p1
https://dmoj.ca/problem/cco15p2

Practise problems:
https://dmoj.ca/problem/persuasion (bitmask dp)
https://dmoj.ca/problem/ccc12s4 (encoding states as integers)
https://dmoj.ca/problem/ccc08s5 (encoding states as integers + game theory)
https://dmoj.ca/problem/dmopc20c6p3 (game theory)
https://dmoj.ca/problem/dpu (bitmask dp)

Basically what we've been doing today but a single part of the state takes multiple bits

Binary literals
In many programming languages (including C++, Java, and Python), you can prefix an integral literal with `0b` to indicate that it is in binary. For example `0b1001011` is the same as 75.

Examples
`0b00110101 << 1` is `0b01101010`