Instituto Politécnico Nacional
Escuela Superior de Cómputo

Inteligencia Artificial

Ejercicio Practico 1

6CV2

Toral Hernández Leonardo Javier

Andres Garcia Floriano
Fecha de Entrega: 04 de septiembre del 2025

Solución propuesta para el problema “The bridge and torch problem”

La solución que propuse para la resolución del problema fue generar todos los posibles movimientos que se pudieran hacerlo haciendo uso de la recursión, además de esto se optimizó de forma que no se volvieran a calcular el mismo estado más de una vez, con el uso de la Programación Dinámica.
Para la representación de los estados del problema, como son originalmente 4 personas que quieren cruzar el puente, entonces para esto use un número binario de 4 bits, cada persona representa un bit, además de que el bit esta apagado si esa persona no ha cruzado todavía el puente y caso contrario el bit esta prendido, aparte de este estado ocupe otro el cual es la cantidad de turnos, en el turno 0 la antorcha está del lado inicial, en el turno 1 está del lado final, en el turno 2 del lado inicial, y así sucesivamente, además de esto se ocupó un limite de rondas porque esto puede ir hasta el infinito si se prueban todas las posibilidades, está esta limitada a 8 rondas.
Para esto se ocupo de la manipulación de bits para saber si estaban prendidos o apagados, como lo es la operación AND para verificar si el i-esimo bit esta prendido, la operación XOR para cambiar el i-esimo bit de prendido a apagado, y la operación OR para prender el bit i-esimo.
El problema es que no guarda de forma explícita los pasos que se usaron para generar la respuesta óptima, lo único que guardan los estados es apartir de ese estado cual es la cantidad mínima que se ocupan para llegar a la solución o en su defecto un valor infinito que significa que no se puede llegar a la solución apartir de ese estado. Por lo cual se necesita de otra función llamada Build(), lo que hace es a partir de los estados, reconstruir la respuesta, simulando todos los posibles movimientos a partir de un estado y elegir el estado que es el óptimo de acuerdo a lo previamente calculado.
De forma que el problema es resuelto pasando por todas las posibles configuraciones de una forma eficiente.
Para la practica se codeo en C++ y en Java.

Código en Java

```java
import java.util.Scanner;

public class practica1 {

    public static int dp[][] = new int[(1 << 4)][8];
    public static int timep[] = new int[4];
    public static int inf = 100000000;

    public static int dpf(int num, int t) {
        if(num == 15)
            return dp[num][t] = 0;
        if(t == 7)
            return dp[num][t] = (num == 15) ? 0 : inf;
        if(dp[num][t] != -1)
            return dp[num][t];
        dp[num][t] = inf;
        if((t & 1) != 0) {
            for(int i = 0; i < 4; ++ i)
                if((num & (1 << i)) != 0)
                    dp[num][t] = Math.min(dp[num][t], timep[i] + dpf(num ^ (1 << i), t + 1));
            return dp[num][t];
        }
        for(int i = 0; i < 4; ++ i) {
            for(int j = 0; j < 4; ++ j) {
                if(((1 << i) & num) != 0 || ((1 << j) & num) != 0) continue;
                if(i == j)
                    dp[num][t] = Math.min(dp[num][t], timep[i] + dpf(num | (1 << i), t + 1));
                else
                    dp[num][t] = Math.min(dp[num][t], Math.max(timep[i], timep[j]) + dpf(num | (1 << i) | (1 << j), t
                }
            }
        return dp[num][t];
    }

    public static void build(int num, int t) {
        if(num == 15) return;
        if((t & 1) != 0) {
            for(int i = 0; i < 4; ++ i) {
                if((num & (1 << i)) != 0 && dp[num][t] - timep[i] == dp[num ^ (1 << i)][t + 1]) {
                    System.out.println(timep[i] + " -> " + timep[i]);
                    build(num ^ (1 << i), t + 1);
                    return;
                }
            }
        }
        for(int i = 0; i < 4; ++ i) {
            for(int j = 0; j < 4; ++ j) {
                if(((1 << i) & num) != 0 || ((1 << j) & num) != 0) continue;
                if(i == j && dp[num][t] - timep[i] == dp[num | (1 << i)][t + 1]) {
```

```
                System.out.println(timep[i] + " -> " + timep[i]);
                build(num | (1 << i), t + 1);
                return;
            }
            if(dp[num][t] - Math.max(timep[i], timep[j]) == dp[num | (1 << i) | (1 << j)][t + 1]) {
                System.out.println("(" + timep[i] + "," + timep[j] + ") -> " + Math.max(timep[i], timep[j]));
                build(num | (1 << i) | (1 << j), t + 1);
                return;
            }
        }
    }
}

Run | Debug
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    for(int i = 0; i < 4; ++ i) {
        timep[i] = scanner.nextInt();
    }
    scanner.close();
    for(int i = 0; i < (1 << 4); ++ i)
        for(int j = 0; j < 8; ++ j)
            dp[i][j] = -1;
    int totaltime = dpf(num:0, t:0);
    System.out.println(totaltime);
    build(num:0, t:0);
}
}
```

Código en C++

```cpp
#include <bits/stdc++.h>

using namespace std;

int dp[(1 << 4)][8];
int timep[4] = {1, 2, 5, 10};
int inf = 100000000;


int dpf(int num, int t) {
    if(num == 15) return dp[num][t] = 0;
    if(t == 7)
        return dp[num][t] = (num == 15) ? 0 : inf;
    int & ans = dp[num][t];
    if(ans != -1) return ans;
    ans = inf;
    if(t & 1) {
        //estan del otro lado
        for(int i = 0; i < 4; ++ i)
            if(num & (1 << i))
                ans = min(ans, timep[i] + dpf(num ^ (1 << i), t + 1));
        return ans;
    }
    for(int i = 0; i < 4; ++ i) {
        for(int j = 0; j < 4; ++ j) {
            if(((1 << i) & num) || ((1 << j) & num)) continue;
            if(i == j)
                ans = min(ans, timep[i] + dpf(num | (1 << i), t + 1));
            else
                ans = min(ans, max(timep[i], timep[j]) + dpf(num | (1 << i) | (1 << j), t + 1));
        }
    }
    return ans;
}

void build(int num, int t) {
    if(num == 15) return;
    if(t & 1) {
        for(int i = 0; i < 4; ++ i) {
            if(num & (1 << i) && dp[num][t] - timep[i] == dp[num ^ (1 << i)][t + 1]) {
                cout << timep[i] << " <- " << timep[i] << '\n';
                build(num ^ (1 << i), t + 1);
                return;
            }
        }
    }
    for(int i = 0; i < 4; ++ i) {
        for(int j = 0; j < 4; ++ j) {
            if(((1 << i) & num) || ((1 << j) & num)) continue;
```

```
            if(i == j && dp[num][t] - timep[i] == dp[num | (1 << i)][t + 1]) {
                cout << timep[i] << " -> " << timep[i] << '\n';
                build(num | (1 << i), t + 1);
                return;
            }
            if(dp[num][t] - max(timep[i], timep[j]) == dp[num | (1 << i) | (1 << j)][t + 1]) {
                cout << '(' << timep[i] << ',' << timep[j] << ") -> " << max(timep[i], timep[j]) << '\n';
                build(num | (1 << i) | (1 << j), t + 1);
                return;
            }
        }
    }
}

int main() {
    for(int i = 0; i < 4; ++ i) cin >> timep[i];
    for(int i = 0; i < (1 << 4); ++ i)
        for(int j = 0; j < 8; ++ j)
            dp[i][j] = -1;
    cout << dpf(0, 0) << '\n';
    build(0, 0);
    return 0;
}
```