Universidad Nacional
de La Matanza

Reconquistando la Tierra de Fantasía

Programación Avanzada

Estudiantes

- DNI: 44.689.133 - Bosch, Maximo Augusto
- DNI: 43.386.520 - Vallejos, Franco Nicolas

Docentes

- Dra. Verónica Aubín
- Ing. Federico Gasior
- Ing. Hernán Lanzillotta
- Ing. Lucas Videla

Comisión

- Jueves- Turno Noche

14 de Noviembre de 2024

Tabla de Contenidos

Resolución

Introducción

El trabajo práctico nos pide que, dado un cierto ejercito, pueblos (aliados o enemigos), y caminos entre los pueblos, determinemos si existe un camino (en lo posible, el más corto posible) entre un pueblo inicial (propio) y un pueblo destino (aliado o enemigo), en donde lleguen tropas vivas (al menos 1 (una)).

Un ejército puede estar formado por tropas de distintas razas, en donde cada tropa tiene un comportamiento individual determinado (dependiendo si descansó en un pueblo aliado o si recibió daño, o cuantas veces atacó antes)

Comportamiento General

Se empieza leyendo un archivo donde están especificados la cantidad de pueblos que hay, que tipo de raza hay en ese pueblo, cuantos habitantes tiene y si es aliado, enemigo o propio. También, se detallan los caminos posibles que existen entre los pueblos, a su vez que el pueblo de inicio y de fin.

Sabiendo todos esos datos, se procede a aplicar el algoritmo de Dijkstra sobre el grafo que representa los pueblos y sus vecinos para obtener el camino más corto entre el pueblo inicial y el pueblo destino. Es importante tener en cuenta que nosotros queremos llegar lo más rápido posible, y si solo nos guiamos por las distancias entre pueblos estamos omitiendo el hecho de que una batalla o un descanso dura un día. Por lo que a la hora de los cálculos, lo que hacemos es “aumentar la distancia” entre los pueblos lo equivalente a lo que recorrería nuestro ejército en ese día de batalla/descanso, en este caso, se nos da el dato de que nuestro ejército recorre 10 km por día, por lo que a cada arista (camino) se le suma a su costo inicial 10 (kilómetros). En caso de que no exista un camino físico entre el pueblo inicio y fin, es en esta parte donde se valida.

Teniendo el camino más corto calculado, se procede a recorrerlo con nuestro ejército, verificando que en cada pueblo nuestro ejército,luego de haber interactuado con un pueblo (batalla o descanso), siga teniendo tropas antes de pasar al siguiente pueblo. Si el ejército sigue teniendo tropas al finalizar el recorrido por ese camino, damos por finalizada la simulación e informamos el estado final del ejército, cuantas tropas quedaron vivas y cuánto tiempo tardó en recorrerlo. En caso que no se llegue por ese camino se aplicará un algoritmo alternativo.

Detalles Técnicos

El comportamiento de las razas y ejércitos se manejan a través del patrón Composite. Se define una interfaz Atacante que indica las acciones que va a poder hacer tanto una Raza

como un Ejército (atacar, recibirDaño, descansar, obtenerVida). Al momento de una batalla o descanso, se acciona sobre un Ejército, el cual replica la acción a todas sus tropas.

Se hizo uso del patrón Template internamente dentro de las razas, haciendo que todos los tipos de razas no emitan daño si su vida es menor a 0 (cero), pero reservando a cada tipo de raza qué hacer cuando ella tenga que atacar, delegando la responsabilidad de cuando efectivamente emitir daño a la clase padre Raza.

Para el mapa se implementó el patrón Singleton, el cual garantiza que solo exista una sola instancia de la clase Mapa durante toda la ejecución del programa, evitando que puedan crearse varios “Mapas” pero con adyacencias cambiadas.

Para obtener el camino más corto se optó por utilizar el algoritmo de dijkstra, ya que este nos permite obtener el camino más corto a todos los nodos con una complejidad de $\theta(v^2)$ donde “v” es la cantidad de vértices (en nuestro contexto, son los pueblos).

Cada pelea tiene una complejidad de $\theta(m + n)$, donde m son la cantidad de tropas aliadas y n la cantidad de tropas enemigas. En cada batalla se van a recorrer las *m* tropas que hayan en un ejército para obtener su daño, y se lo va a aplicar a las *n* tropas del otro ejército (en el peor caso, que es cuando muere todo el ejército). Si bien no sabemos cuantas veces se va a repetir ese proceso para una pelea, podemos tomarla como una constante. Genéricamente se recorrerán *k*(m+n)* veces las tropas, siendo *k* un número natural, el cual no influye a la hora de dar una cota superior

Punto extra

Para encontrar el camino más corto donde el ejército sobreviva es inevitable el hecho de tener que recorrer todos los caminos posibles desde el nodo inicial hasta el final, buscando caminos donde el ejército sobreviva y quedándonos con los de menor costo. Pero el hecho de tener que recorrer todos los caminos posibles implica una complejidad de $\theta(v!)$, lo cuál hace inviable el uso de este algoritmo. Por ende, optamos por utilizar programación dinámica para aliviar la necesidad de tener que recorrer caminos más de una vez.

La cuestión es, ¿Cómo aplicamos programación dinámica? Una de las variables necesarias es un ejército donde la probabilidad de encontrar dos ejércitos exactamente iguales en un mismo nodo, que pasó por distintos caminos, es casi cero. Por esto mismo utilizaremos la siguiente estrategia donde utilizamos una memoria auxiliar para guardar resultados parciales, pero utilizaremos estos resultados si cumplimos ciertas condiciones.

Imaginemos un escenario donde existen dos caminos posibles, c1 y c2, que convergen en el mismo nodo A, y este nodo tiene como único vecino al nodo destino B.

Al recorrer por primera vez los caminos, llegamos al nodo A a través de c1 y luego avanzamos hacia el nodo destino B. Si el ejército que llegó a B a través de c1 es derrotado, registramos en A un "resultado parcial", que contiene la información sobre el ejército con el

que llegamos al nodo A, el camino recorrido hasta el destino (o hasta el nodo donde murió) y el desenlace del encuentro (derrota o victoria), usando el nodo A como clave.

Más adelante, al llegar al nodo A por el camino c2, encontramos que ya existe un resultado parcial registrado en A. Ahora, comparamos el ejército con el que llegamos a A a través de c2 con el ejército del resultado parcial guardado:

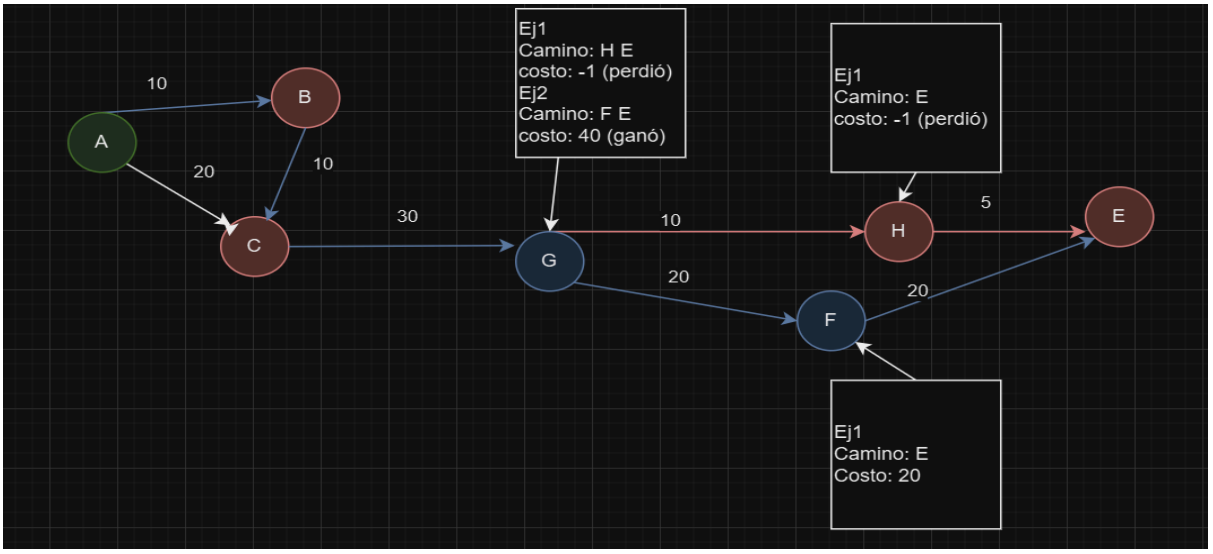
- Si el ejército de c2 es inferior al del resultado parcial y sabemos que el ejército del resultado parcial no alcanzó el destino, podemos concluir que el ejército de c2 también fracasaría al tomar el mismo camino. En este caso, descartamos c2 y buscamos otras alternativas.
- Si el ejército de c2 es superior al del resultado parcial y este no alcanzó el destino, entonces no tenemos suficiente información para deducir el desenlace. Esto nos obliga a recorrer el camino desde A hasta el destino para determinar si el ejército de c2 tiene éxito.

¿Y en el caso de que c1 salga victorioso? Sólo cambiará nuestro criterio de decisión:

- Si el ejército de c2 es superior al del resultado parcial y sabemos que el ejército del resultado parcial alcanzó el destino, podemos concluir que el ejército de c2 también llegará al destino al tomar el mismo camino. En este caso, no seguimos por c2 y utilizamos el resultado parcial con su costo.
- Si el ejército de c2 es inferior al del resultado parcial y este alcanzó el destino, entonces no tenemos suficiente información para deducir el desenlace. Esto nos obliga a recorrer el camino desde A hasta el destino para determinar si el ejército de c2 tiene éxito.

A su vez también puede existir el caso donde el nodo A no tiene camino hasta el nodo B. En este caso se guardará en memoria que el nodo A es un camino sin salida para que en futuros recorridos no hacer el mismo camino.

A continuación mostraremos ejemplos de cómo actuaría este algoritmo en cada recorrido, marcando en azul los nodos que representan los pueblos aliados, y en rojo los nodos que representan los pueblos enemigos. También se pintaron las aristas representando el camino que hizo el algoritmo, marcando con rojo las derrotas y con azul las victorias.



Las etiquetas representan lo guardado en memoria para dicho nodo. En este caso vemos el recorrido que hizo el algoritmo desde A -> B -> C -> G (c1) para luego bifurcarse y hacer: G->H->E (c2) y G->F->E (c3).

Vemos que en el camino c2 el ejército muere en el nodo E (final). Por ende el nodo H se guarda que el ejército no llegó al destino por el camino E. En este mismo camino tenemos que el nodo H no llega al destino, por esto mismo el nodo G guarda el ejército con el camino H->E indicando que no llega al destino.

Ahora al ver el camino c3, vemos que el ejército llega al destino conquistándolo, entonces su predecesor F guarda que el ejército llegó al destino indicando el camino E y el costo. A su vez vemos que el nodo F llega al destino, por esto mismo el nodo G guarda el ejército con el camino F->E indicando su costo.

Ahora la duda es, ¿Que guardo en el nodo C?

El nodo G al tener un camino donde se llega al destino y un camino donde no, no puede decirle con certeza al nodo C cuáles son las condiciones de victoria y derrota. Estos casos son donde no podemos seguir guardando los resultados parciales y debemos llegar a G para poder estimar los resultados. Tener una victoria y una derrota significa que puede existir un camino alternativo donde no se llegó que pueda tener un menor costo que el camino que marcó la victoria.

En los próximos recorridos se utilizará la información guardada para intentar estimar los resultados. En caso de no poder hacerlo, se verá obligado a tener que cumplir el recorrido.

Se llegó a esta solución ya que, por la naturaleza de las tropas, no es posible predecir cómo va a ser una batalla teniendo los datos de los ejércitos, ya que cada batalla es única porque depende del estado interno de cada tropa individual. Por lo que no es tan fácil como solamente aplicar un “costo” extra en cada nodo que representaría la fuerza/vida de un pueblo y aplicar

Conclusiones

La complejidad computacional de este programa, quitando el punto extra y la carga del archivo inicial, es igual a $\theta(v * (m + n))$, asumiendo que *m+n > v*. Eso incluye el cálculo del camino más corto usando Dijkstra $\theta(v^2)$ mas el costo de hacer tantas batallas como pueblos haya (en el peor caso) $\theta(v * (m + n))$. Por lo que si asumimos que *m+n > v,* la complejidad final se la lleva el hecho de hacer las batallas (por regla de la suma).

Este trabajo práctico nos hizo repasar y reforzar los conceptos de POO (encapsulamiento, herencia y polimorfismo) y sobre cómo la programación dinámica puede ayudar a resolver problemas que a priori la única solución sería la fuerza bruta.

Bibliografía

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Patrones de diseño: Elementos de software orientado a objetos reutilizable. Addison-Wesley. https://profeuttec.yolasite.com/resources/Patrones%20de%20dise%C3%B1o%20-%20Erich%20Gamma.pdf
- Navona, E. (2024, Oct 24). Algoritmo de la ruta más corta de Dijkstra: Introducción gráfica y detallada. FreeCodeCamp. https://www.freecodecamp.org/espanol/news/algoritmo-de-la-ruta-mas-corta-de-dijkstra-introduccion-grafica/