Victoria González Chacín – 202320852

Zair Montoya – 202321067

Andrea Dávila –

El problema de “Samus y el laberinto de Zebes” es un problema que puede ser planteado de manera efectiva a través de un grafo y algoritmos de búsqueda en este. De esta forma estructuramos nuestra respuesta siguiendo los siguientes pasos:

1. Entender e identificar las características principales de la problemática.
2. Estructurar y ver las restricciones del grafo a usar.
3. Establecer el algoritmo de búsqueda que usaremos.

Para la primera parte tuvimos en cuenta las siguientes consideraciones en el juego de Metroid:

1. Va a existir un orden en donde la plataforma 0, o la plataforma en donde Samus se encuentra al inicio del juego, será la inicial, y la plataforma n-1 siempre será nuestra plataforma final o destino, o en donde se encontrará Raven Beak.
2. Samus puede saltar en cualquier dirección (para atrás o adelante) de una plataforma a la siguiente.
3. En ciertas plataformas hay poderes que le permite a Samus dar un salto de k plataformas, también en cualquier dirección.
4. Samus tiene la capacidad de teletransportarse en cualquier momento (por lo tanto, en cualquier plataforma), a la plataforma que desee (que consume el número de plataformas por las cuales se mueva de energía).
5. En ciertas plataformas hay robots asesinos, que no pueden ser usadas.

De esta forma empezamos a desarrollar nuestra solución. Aun así, es importante aclarar desde un inicio las distintas formas a las cuales llegamos a plantear una misma problemática, viendo como hacer la siguiente aún más eficiente en cuanto a memoria y tiempo de ejecución. La mayor clarificación que tuvimos en cuenta fue la magnitud de plataformas que puede haber, definida como 2<=n<=10^5. Es por esto por lo que planteamos que el peor caso seria un grafo en donde se tienen 100,000 plataformas, no hay energía para teletransportarse, y no hay saltos o poderes disponibles, si no que Samus únicamente puede moverse para adelante o para atrás en plataformas adyacentes.

**Construcción del grafo:**

De esta forma sabemos que los nodos serán las n plataformas existentes, y las aristas serán las conexiones que se dan cuando Samus puede saltar a las plataformas subsiguientes, cuando puede usar poderes para saltar a la plataforma actual + k saltos en cualquier dirección, y cuando quiera teletransportarse de su posición actual a cualquier otra plataforma. El momento definitivo en donde no van a haber conexiones entre plataformas es cuando hay robots asesinos en una de ellas, ya que esto haría automáticamente que Samus pierda el juego. También es importante aclarar como Samus no empieza directamente en la plataforma numero uno, sino que es necesario que el salte a esta como primer movimiento para así empezar el recorrido de n plataformas, es por esto por lo que el primer paso siempre ser C+. Y de esta posición solo podrá saltar para adelante, no podrá teletransportarse, ni se le asignará ningún poder.

Ya con el grafo construido definimos que el problema que se busca solucionar es encontrar el camino más corto (menor número de movimientos) desde la plataforma 0 hasta la plataforma n-1, siempre que tenga suficiente energía para hacerlo.

Aun así, en un primer planteamiento de la solución se notó como únicamente la creación del grafo resulta en un gran uso de tanto tiempo como de memoria durante la ejecución, por lo que siempre resulta en un grafo altamente conexo y completo. Continuamente se analizo que independientemente del algoritmo que se eligiera este se vería obligado a hacer la búsqueda del camino mínimo a través de cada vértice existente, lo cual, en el peor caso, resultaría en una búsqueda masiva y densa. De esta forma propusimos no tener un método que creara un grafo, sino implementar un único método que uniera la búsqueda con la creación del grafo, de esta forma solo se buscaría y se crearan los vértices en caminos efectivos, descartando muchas posibilidades que no llevan directamente a la respuesta que queremos.

**Algoritmo seleccionado:**

Teniendo en cuenta que tenemos aristas de igual costo y que solo nos interesa el minimizar el número de movimientos (no funcionaría DFS, ya que no garantiza el camino más corto), no el peso de las acciones (por lo que no nos serviría Dijsktra), podemos usar el algoritmo de BFS para explorar primero los caminos más cortos, no atascarnos en caminos más largos innecesarios, evitar ciclos y caminos inválidos (considerando que nuestro grafo puede tener muchos caminos posibles).

**Creación de la solución:**

Se sigue el algoritmo estándar de bfs, iniciando con un set de plataformas visitadas y una cola con una tripleta que contiene la plataforma de inicio (0), la energía que tiene Samus y una lista vacía para ir llenándola con los pasos de cada recorrido. De la misma forma se define las plataformas que tienen robots, y un diccionario que contiene los poderes, organizándose por plataforma de inicio, y numero de saltos. Mientras haya elementos por explorar en la cola, usando un while, tomamos el siguiente nodo a explorar (recordando la tripleta que guarda la información de la plataforma actual, energía restante y el camino recorrido hasta ese punto), haciendo un pop de la cola. Después verificamos que, si llegamos a la última plataforma, o en la que esta Raven Beak, devolvemos el camino, ya que significaría que terminamos el recorrido. Por otro lado, si ya estuvimos en esta plataforma con esa energía exacta (posición actual, energía restante), ignoramos este vértice ya que esa combinación exacta ya fue explorada previamente y volver a procesarla no aportara un camino más corto, solo haciendo el algoritmo más redundante, y marcamos el estado como visitado.

Por consiguiente, empezamos la búsqueda y “construcción” formal del grafo. Recorremos todos los movimientos o acciones posibles desde la plataforma actual.

De esta forma comenzamos con el proceso de construcción para los movimientos de caminar (C+ y C-):

* Ya que Samus puede caminar para adelante y hacia atrás hacemos dos verificaciones; si la siguiente plataforma (plataforma actual+1) no se sale de los rangos (0<n<n-1) y tampoco tiene robots entonces podemos añadir esa conexión a la cola guardando la siguiente información en la tripleta (posición actual + 1, energía restante, acciones), y lo mismo para la plataforma anterior (posición actual-1) con (posición actual - 1, energía restante, acciones).

Seguimos con los poderes de saltos (S- y S+):

* Empezamos verificando que la posición actual este dentro de las plataformas que tienen poderes, se revisa el salto, y si la posición actual + salto (S+) (el destino) está dentro del rango y no tiene robots, añadimos esa conexión a la cola, y lo mismo para posición actual – salto, añadiendo (posición actual - salto, energía restante, acciones + ["S-"]).

Terminamos con los momentos en donde Samus pueda requerir teletransportarse (Tx):

* Recorremos todas las plataformas a las cuales Samus podría teletransportarse (k), las cuales deben de estar en el rango de la energía disponible, por lo que no puede teletransportarse a lugares en donde la energía no alcanza. También recordamos que desde la plataforma 0 Samus no se puede teletransportar, ni se puede hacerlo a esta. De esta forma también se reduce la cantidad de conexiones posibles, ya que mientras se itera por cada conexión se va reduciendo la energía, consecuentemente reduciendo la cantidad de destinos posibles. Se calcula cual seria la plataforma destino, viendo si se movería para adelante o atrás, y si esta no tiene robots y sigue en el rango de posibilidades, entonces se guarda en la cola, restándole a la energía restante la cantidad de saltos que se tendrían que hacer, con esto queda (destino\_tp, energia\_restante - k, acciones + [accion]).

Si terminamos de explorar todo y no llegamos a la plataforma final, no hay camino valido, y devolvemos “NO SE PUEDE”.

Forma, Círculo

El contenido generado por IA puede ser incorrecto.

**Complejidad:**

La complejidad temporal del algoritmo está determinada principalmente por dos funciones: construccion\_grafo y BFS. En la función construccion\_grafo, se itera sobre cada plataforma para construir las conexiones válidas. Esta función contiene un doble bucle anidado en el que por cada plataforma i, se evalúan todos los posibles destinos j con los que podría conectarse mediante un salto "T". En el peor caso, si no hay robots que bloqueen caminos y todos los saltos son válidos, se generan hasta n² conexiones, resultando en una complejidad temporal de O(n²) para la construcción del grafo. También se evalúan los poderes especiales (acciones "S+” y “S-”), que en el peor caso agregan una complejidad adicional de O(n), aunque esta no cambia el orden final.

Por su parte, la función BFS explora el grafo generado utilizando una cola que almacena tuplas de estado con la forma (plataforma\_actual, energia\_restante, camino\_corrido). La clave aquí es que el espacio de estados no está dado solo por la plataforma actual, sino también por la energía restante, lo que implica que podrían existir hasta n \* energía combinaciones únicas que deben visitarse. Por ello, la complejidad temporal del BFS en el peor de los casos puede alcanzar O(n \* energía \* d), donde d es el grado de cada nodo (es decir, el número de acciones posibles desde una plataforma). En un grafo completamente denso, donde cada plataforma puede conectarse con casi todas las demás, este grado puede ser O(n), llevando a una complejidad temporal de O(n² \* energía) para esta fase.

En cuanto a la complejidad espacial, se deben considerar varias estructuras. El grafo construido puede ocupar hasta O(n²) espacio si todas las conexiones posibles se almacenan. La cola usada por BFS podría llegar a contener hasta O(n \* energía) elementos en un escenario donde se exploren muchas combinaciones de plataforma y energía antes de alcanzar la meta. El conjunto visitado también podría almacenar hasta O(n \* energía) estados únicos. Además, cada entrada en la cola de BFS lleva consigo una copia del camino recorrido hasta ese punto, y aunque en muchos casos los caminos son cortos, en el peor caso podrían ocupar espacio O(n) cada uno, lo que puede llevar a un uso total de memoria de hasta O(n²) si no se optimiza adecuadamente esta estructura.

En resumen, este algoritmo presenta una complejidad temporal de O(n² + n² \* energía), resultado de la combinación de la construcción completa del grafo y la exploración exhaustiva de estados en la búsqueda. En cuanto a la complejidad espacial, se puede estimar como O(n² + n \* energía) por el almacenamiento del grafo, la cola de BFS, el conjunto de visitados y los caminos parciales. Este enfoque es adecuado para valores moderados de n y energía, pero podría ser mejorado para casos más exigentes evitando la construcción completa del grafo y generando dinámicamente las transiciones durante la exploración.