TALLER N°1

Problema del Granjero y el río (general)

usach

Taller de programación 1-2023

Fecha: 13 de Abril

Autor: Sergio Espinoza

TALLER N°1

Problema del granjero y el río (general)

Explicación breve del algoritmo

En general se leen los datos del archivo, la mayoría lo guarda directamente, a excepción de las restricciones las cuales se guardan en una matriz que sigue la forma restricciones x ítems (donde los ítems son los ítems en sí más los granjeros). Luego de leer los datos empezamos con la solución general, en la cual se hace uso de un bote que se va llenando revisando las restricciones y todo, luego se cruza al lado derecho y se revisa si finalizó (como solo puede terminar al pasar al lado derecho, revisaremos si finalizó el algoritmo en estos casos, más no cuando el bote cruce al lado izquierdo; si no ha terminado se cruza lo mínimo hacia el lado izquierdo y se vuelve a empezar hasta que estén todos de un lado o ya no haya nada en el stack de abiertos.

Heurísticas o técnicas utilizadas

La heurística general usada fue el método goloso, donde del lado izquierdo al derecho se lleva la mayor cantidad de ítems posibles, en cambio de derecha a izquierda se lleva el mínimo (de ser posible solo un driver).

Funcionamiento del programa

En main

- Se revisa que se haya ingresado el nombre del archivo como argumento.
- Se inicializa el objeto con el juego.
- Se cargan los datos del archivo (si no se logra abrir se avisa y se detiene el programa).
- Se resuelve el juego.
- Se cierra y termina el programa.

En game

- Para la lectura y carga de datos:
    - Se abre el archivo.
    - Se lee la primera línea y se almacenan los datos de drivers, items, espacio bote (también se almacena el valor n = drivers + items).
    - Se obtiene el número de restricciones izquierdas y se genera la matriz de restricciones con valores “false”.
    - Se leen las restricciones y se actualizan sus valores en la matriz por “true”.
    - Se repite el mismo proceso con las restricciones de la derecha.
- Para resolverlo:

- Se inicializa el estado inicial (todo “true” a la izquierda y “false” a la derecha) y se agrega al stack de abiertos.
- Se inicializa el bote (vacío).
- Mientras el stack de abiertos no este vacio:
  - Cruce a la derecha:
    - Se agrega al bote la mayor cantidad de objetos posibles teniendo en cuenta las restricciones asociadas, y luego se cruza el río.
    - Se agrega el estado al stack de visitados y el nuevo al de abiertos.
  - Se verifica si ya finalizó, en dicho caso se termina.
  - Cruce a la izquierda:
    - Se agrega un único driver (el primero si hay más de uno) y se verifica si se puede devolver, si lo puede hacer se devuelve sólo, si no, se devuelve con el mínimo de ítems posible.
    - Se agrega el estado al stack de visitados y el nuevo al de abiertos.
- Si se llega al stack de abiertos vacío y no se encontró la solución entonces se avisa que no se encontró una solución posible al problema y se termina.

Aspectos de implementación y eficiencia

Respecto a la eficiencia, el método goloso es eficiente en término de tiempo pero no necesariamente eficaz, esto debido a que no es tan efectivo a la hora de encontrar la solución óptima sino una buena solución. El problema está en la revisión de restricciones donde se hace con un doble bucle lo cual es muy ineficiente; lo que se trata de reducir no revisando los casos donde el ítem que se revisa no está en dicha restricción.

Ejecución del código

En sí, no logre terminar el código por lo que no daré instrucciones para ejecutarlo debido a que no se puede.

En caso de que hubiese estado listo hubiese bastado con un “make” y “./bin/main NombreArchivo.txt” para compilarlo y ejecutarlo.