PRÁCTICA 2
ALGORITMIA BÁSICA

Diego Martínez Baselga 735969
David Solanas Sanz 738630

En esta práctica se ha desarrollado un algoritmo de ramificación y poda que, dada una lista de pedidos de estaciones individuales en la línea Taskent-Samarcanda, calcula el máximo ingreso total para la compañía uzbeka. Se han desarrollado los ficheros Peticion.hpp, NodoPeticion.hpp y transporte.cpp, que constituyen el código fuente; junto con un fichero Makefile. Se ha hecho uso de las clases predefinidas vector<T> de la biblioteca <vector> y priority_queue<T> de la biblioteca <queue> de C++ para la gestión de vectores y de la cola de prioridades de los nodos vivos. También se ha hecho uso de la biblioteca <algorithm> para la función sort para ordenar el vector de pedidos de reserva.

Peticion.hpp

En primer lugar, se ha desarrollado el fichero Peticion.hpp, que implementa el tipo abstracto de dato correspondiente a un pedido de reserva de viaje. El TAD tiene los siguientes atributos:

- Estacion_salida (entero): Almacena la estación de salida del pedido.
- Estacion_llegada (entero): Almacena la estación de llegada del pedido.
- Num_pasajeros (entero): Indica el número de pasajeros que incluye el pedido.

Tiene 2 constructores distintos. Si se llama al constructor sin parámetros se crea un pedido con todos los parámetros inicializados al valor -1. Si se le llama con 3 enteros (salida, llegada y pasajeros), se crea el pedido con los valores especificados en los parámetros de entrada.

Tiene 2 funciones (además de sus correspondientes setters y getters), que no reciben ningún parámetro, el primero, “getBeneficio” calcula el beneficio obtenido con ese pedido cambiado de signo, el segundo, “getDistancia” calcula la distancia del trayecto reservado. Se ha definido también el operador de comparación entre dos pedidos, un pedido será menor que otro si el número de pasajeros / distancia del trayecto es menor que el de otro pedido. Se ha elegido esta comparación porque al ordenar las peticiones según este criterio se reduce en mayor medida el tiempo de cómputo del algoritmo que sin ordenar las peticiones u ordenándolas por beneficio.

NodoPeticion.hpp

Tipo abstracto de dato utilizado para representar un nodo del árbol de soluciones del problema. Tiene los siguientes atributos:

- Capacidad_ocupada (vector<int>): Indica la capacidad ocupada de cada estación, el tamaño del vector viene determinado por el número total de estaciones.
- Coste (entero): Almacena el valor de la función de cota del nodo
- Estimación (entero): Almacena el valor de la función de poda del nodo
- Id_petición (entero): Índice que corresponde al i-ésimo pedido del conjunto de pedidos de reserva.
- Descartados (vector<bool>): Indica que peticiones han sido descartadas (true)

Tiene 1 constructor, al cual se le pasan 3 parámetros enteros (id, size, n_paradas) y se crea un nodo con id_petición [id], inicializa el vector de capacidad ocupada a 0 de [1..n_paradas] e inicializa el vector de descartados a false de [1..size]. Tiene un procedimiento (además de sus correspondientes getters y setters) llamado “descartar” el cual recibe un parámetro (i) y marca el pedido i-ésimo como descartado.

Transporte.cpp

El fichero transporte.cpp contiene la función main que corresponde con el programa principal. Comprueba si los parámetros introducidos son correctos y si el usuario ha introducido el flag "-fb" como **último** parámetro (**opcional**). Este último parámetro hace que el problema se resuelva utilizando un algoritmo de fuerza bruta en lugar del algoritmo de ramificación y poda, esto se ha hecho para la comprobación de la corrección de los resultados generados por el algoritmo de ramificación y poda, si no se añade el flag "-fb" al final se ejecuta el algoritmo de ramificación y poda. El algoritmo que resuelve un problema sigue el siguiente esquema:

- Leer el fichero de datos: Se obtienen los datos del problema con la función obtenerDatos, que extrae del fichero los datos y devuelve un vector de pedidos de reserva (vector<Peticion>), un entero (n) que corresponde a la capacidad total del tren y otro entero (m) que corresponde al número total de estaciones del problema. Si quedan más datos correspondientes a distintos problemas devuelve true, si ya no quedan más problemas devuelve false.
- Ordenar vector de pedidos: Ordena el vector de pedidos obtenido al leer del fichero según el criterio descrito anteriormente para la clase Petición, de mayor a menor.
- Iniciar el algoritmo: Se llama a la función calcular_solución, que, en primer lugar, crea el nodo raíz del espacio de soluciones y calcula su valor de cota y de poda.
- Función cota: Para calcular el valor de cota se hace uso de la función función_cota que recibe como parámetros el vector de pedidos, el índice de el pedido actual (objeto), la capacidad total del tren, un vector de booleanos de pedidos descartados y un vector de enteros de la capacidad ocupada en cada estación. Calcula el beneficio por los pedidos que no han sido descartados, si se encuentra algún pedido que no caben todos los pasajeros, se rellena el tren por completo con todos los pedidos no descartados restantes actualizando el valor del beneficio.
- Función poda: Para calcular el valor de poda (equivalente al beneficio obtenido cambiado de signo) se hace uso de la función función_poda que recibe como parámetros el vector de pedidos, el índice de el pedido actual (objeto), la capacidad total del tren, un vector de booleanos de pedidos descartados y un vector de enteros de la capacidad ocupada en cada estación. Calcula el beneficio por los pedidos que no han sido descartados, si se encuentra algún pedido que no caben todos los pasajeros se ignora dicho pedido y se pasa al siguiente.
- Ramificación y poda: Una vez creado el nodo raíz del espacio de soluciones, se llama a la función ramificación_poda que en primer lugar crea una cola de prioridades vacía para guardar los nodos del espacio de soluciones y añade el nodo raíz, a partir de ese momento ejecutará mientras haya algún nodo en la cola el siguiente algoritmo.
- Desapilar nodo más prometedor: Se obtiene el nodo cuyo coste sea mínimo y se quita de la cola de prioridades.

- Comprobar si hay que podar o no: Si el coste del nodo desapilado supera el valor mínimo de poda se realiza una poda, es decir, no se generan los nodos hijos del nodo actual ni se añade a la cola de prioridades y se vuelve a desapilar el siguiente nodo más prometedor. **IMPORTANTE**: Si el usuario había especificado al ejecutar el programa el flag "-fb" esta condición es ignorada y nunca se realizará la poda.
- Expandir nodo: Si no ha habido poda se expande el nodo actual generando su hijo izquierdo, que corresponde al nodo generado cuando se escoge el pedido actual para la solución y su hijo derecho, que corresponde al nodo generado cuando no se escoge el pedido actual para la solución. Para el hijo izquierdo se comprueba si es un nodo factible, si al añadir el pedido no se supera el límite de capacidad del tren, si se supera dicho nodo será descartado y no se tendrá en cuenta para el cálculo de la solución.
- Actualizar valores de cota, de poda, capacidad y pedidos descartados de los hijos: Para los hijos izquierdos que no superen el límite de capacidad del tren, el valor de cota y poda será el mismo que el de su padre, su vector de capacidades será actualizado sumándole para cada estación del pedido el número de pasajeros del pedido y el vector de pedidos descartados será el mismo que el del nodo padre. Para los hijos derechos, el vector de capacidades será el mismo que el del nodo padre (puesto que no se añade ningún pedido), se actualizará el vector de descartados, descartando el pedido actual y se calculará su valor de poda y cota con las funciones función_poda y función_cota explicadas anteriormente.
- Comprobar si hijos son hojas o nodos intermedios del árbol: Si los nodos expandidos son nodos intermedios, se añaden a la cola de prioridades para ser expandidos más adelante, si el nodo izquierdo expandido superaba el límite de capacidad del tren al añadir el pedido actual no será añadido a la cola de prioridades. Si los nodos expandidos son hojas, si el hijo izquierdo no es factible se ignora y se comprueba el hijo derecho, en caso contrario se ignora el hijo derecho y se comprueba el hijo izquierdo.
- Actualizar valor mínimo de poda (a su vez máximo beneficio obtenido): Para las soluciones factibles (nodos hoja, izquierdo o derecho dependiendo de la comprobación anterior), si el valor de coste del nodo es inferior al valor mínimo de poda, dicho valor se actualiza al valor de poda del nodo con coste inferior.
- Repetir proceso (desapilar siguiente nodo más prometedor) hasta que no queden nodos vivos en la cola de prioridades: Una vez la cola de prioridades quede vacía la variable U contendrá el valor mínimo de poda que coincide con el beneficio máximo obtenido cambiado de signo, la función devolverá dicho valor.
- Almacenar valor en el fichero de salida: El valor devuelto por el algoritmo será escrito en el fichero de salida junto con el tiempo de ejecución del algoritmo.

Pruebas

Las pruebas realizadas han sido ejecutar las 8000 instancias del problema con un algoritmo de fuerza bruta (flag "-fb") y con el implementado, y se ha comprobado que los beneficios obtenidos en ambos casos coinciden para cada instancia del problema.