PRÁCTICA FINAL ADA

RAMIFICACIÓN Y PODA

Marcos Cerdán Amat 20518142-A

---

1. Estructuras de datos

1.1 Nodo

Un nodo es un punto de intersección entre otros elementos. En esta práctica, empezamos el con el nodo inicial vacío y lo iremos expandiendo según si tomamos un valor (1) o no lo tomamos (0).
Con estos criterios iremos generando un árbol con todo el espacio de soluciones formado por estos nodos.

```
for (unsigned j = 0; j<2; j++){ //Expandimos los hijos de un nodo
    x[k] = j;
    int new_value = value + x[k] * v[k]; //Calculamos los valores de los nodos
```

1.2 Lista de nodos vivos

La lista de nodos vivos se compone por los nodos que son prometedores, es decir los que no han sido podados y cumplen con las restricciones del problema y en un futuro servirán para generar una posible solución.
Estos nodos se irán añadiendo a una cola de prioridad.

```
if(opt_bound > best_val){ //Si el valor óptimo es mayor que la cota pesimista
    Nodos_vivos++;
    pq.emplace( opt_bound, new_value, x, k+1); //Añadimos el nodo a la cola de prioridad
}
```

2. Mecanismos de poda

2.1. Poda de nodos no factibles

Una poda sería podar los nodos cuya solución no cumplan con alguna de las restricciones, en este caso cuando un valor supere el tamaño máximo de la mochila pasará a ser un nodo no factible y se podará

```
    if(new_value <= V){ //Tiene que cumplir la restricción de que el nuevo valor no desborde la mochila
```

2.2. Poda de nodos no prometedores

Un mecanismo sería fijarnos en que el valor de ese nodo no sea inferior a la cota pesimista, de serlo, también se podaría puesto que no obtendríamos un mejor resultado del que ya tenemos

```
if(opt_bound > best_val){ //Si el valor óptimo es mayor que la cota pesimista
```

3. Cotas pesimistas y optimistas

3.1 Cota pesimista inicial (inicialización)

La cota pesimista inicial la inicializaremos haciendo uso de un algoritmo voraz de la mochila discreta

```
int cota_pesimista(const vector<int> v, size_t k, int V){

    int acc_v = 0;

    for ( unsigned i = k; i < v.size() ; i++){
        if( v[i] < V ) {
            acc_v += v[i];
            V -= v[i];
        }
    }
    return acc_v;
}
```

3.2 Cota pesimista del resto de nodos

La cota optimista se irá actualizando según se encuentre una mejor solución para tener esta como referencia.

Esta será la mejor entre la que ya teníamos y una mejor que encontremos.

```
best_val = max(best_val, new_value + cota_pesimista(v, k+1, V- new_value));
```

3.3 Cota optimista.

Una buena cota optimista que he encontrado ha sido:

Si tengo una mochila de 100 y ya he llenado 98, si el siguiente objeto de la mochila pesase 10, el valor optimo es el tamaño de la mochila, es decir, opt_value = 100.

```
int cota_optimista(const vector<int> &v, size_t k, int V){

    int acc_v = 0.0;

    for( unsigned i = k; i < v.size() ; i++){
        if( v[i] > V ) {
            acc_v += V;
            break;
        }
        acc_v += v[i];
        V -= v[i];
    }
    return acc_v;
}
```

4. Otros medios empleados para acelerar la búsqueda

Una forma de acelerar la búsqueda es quedarnos con una sola solución óptima, es decir, en el momento que encontremos una solución que sea insuperable, nos quedaremos con esa solución y no buscaremos más

```
if(value == V){ //si el valor actual que tenemos es el óptimo, dejamos de buscar
    continue;
}
```

5. Estudio comparativo de distintas estrategias de búsqueda

En esta práctica he podido comprobar que organizar los elementos antes de su búsqueda no siempre es bueno, puesto que si por ejemplo ordenamos ascendentemente los valores y resulta que la solución óptima se encuentra en los 3 últimos objetos, estaríamos recorriendo todo el vector de forma innecesaria, por eso no podemos saber si organizar el vector es siempre una buena solución.

6. Tiempos de ejecución

Fichero 1 bb.problem: 0.071 ms.
Fichero 10 bb.problem: 0.198 ms.
Fichero 20 bb.problem: 3.627 ms.
Fichero 30 bb.problem: 3552.98 ms.
Fichero 40 bb.problem: 23602.7 ms.
Fichero 45 bb.problem: 2122.63 ms
Fichero 50 bb.problem: 2.118 ms.
Fichero 100 bb.problem: 1.945 ms.
Fichero 200 bb.problem: 11.672 ms.
Fichero 1k bb.problem: 109.817 ms.
Fichero 2k bb.problem: 83.233 ms.
Fichero 3k bb.problem: 158.475 ms.