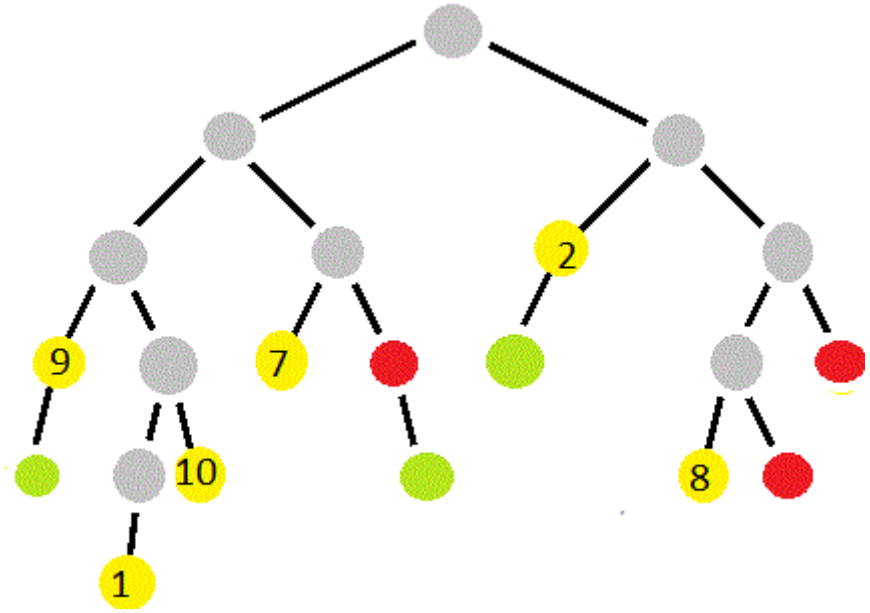
RAMIFICACIÓN Y PODA
PRÁCTICA FINAL

SUMA MÁXIMA LIMITADA (IV)



Análisis y Diseño de Algoritmos – 2019/2020

Grupo de prácticas – 02

INDICE

1. Estructuras de datos

```
typedef vector<int> Sol;
//         opt_bound, value, x, k
typedef tuple<int, int, Sol, unsigned> Node; // ponemos
priority_queue< Node > pq; // priority queue = max-heap
```

1.1 Nodo

El nodo está compuesto por el valor de cota óptima, el valor, un vector solución para almacenar los valores factibles y una variable k para ir revisando los valores que tenemos e ir identificando donde se encuentra el valor que nos interesa.

1.2 Lista de nodos vivos

Uso de una cola de prioridad donde se van almacenando los nodos vivos. He introducido el valor de cota óptima al inicio de la tupla para que la cola de prioridad ordene por este valor. Cuando el valor óptimo es mejor que el que tenemos lo almacena.

2. Mecanismos de poda

```
if(value == T) continue; // Si tenemos el valor máximo dejamos de buscar.
```

```
if(new_value <= T){ // is feasible
    //nº de veces que le mejor solución hasta el momento se ha actualizado a pa
    int actualMejor = best_val;
    best_val = max(best_val, new_value + knapsack_d(sorted_v,k+1,T-new_value));
    if(actualMejor != best_val){
        current_best_updates_from_pessimistic_bounds++;
    }

    int opt_bound = new_value + opt_b(sorted_v,k+1,T-new_value);

    if(opt_bound > best_val){ // is promising
        // nº de nodos añadidos a la lista de nodos vivos.
        pq.emplace(opt_bound, new_value, x, k+1);
        added_nodes++;
    }
```

2.1 Poda de nodos no factibles

Si el nuevo valor es mayor que el valor máximo, entonces se descartan los nodos ya que no son factibles.

2.2 Poda de nodos no prometedores

Si el valor de la cota óptima es menor o igual a la mejor solución hasta el momento, entonces el nodo no es prometedor.

```
int opt_bound = value + opt_b(v,k+1,T-value);
if(opt_bound <= best_val){ // if it's not promising
    discarded_promising_nodes++; //nº de nodos que fueron prometedores,
}
```

3. Cotas pesimistas y optimistas

3.1 Cota pesimista inicial (inicialización)

La cota pesimista inicial se inicializa usando la mochila discreta mediante un algoritmo voraz.

```
int best_val = knapsack_d(v, 0, T);
```

3.2 Cota pesimista del resto de nodos.

Una vez inicializado la cota pesimista a la variable best_val, ésta va actualizando la cota en la misma variable.

```
best_val = max(best_val, value);
```

3.1 Cota optimista.

Inicialmente se inicializa la cota optimista mediante un algoritmo voraz.

```
priority_queue< Node > pq; // priori

int best_val = knapsack_d(v, 0, T);
int opt_bound = opt_b(v, 0, T);
```

4. Cotas pesimistas y optimistas.

He utilizado el siguiente mecanismo para acelerar la búsqueda:

Si el valor es igual al valor máximo entonces dejamos de buscar, ya que hemos encontrado la solución óptima.

```
if(value == T) continue; // Si tenemos el valor máximo dejamos de buscar.
```

Por un lado, si no utilizo el mecanismo anterior para acelerar la búsqueda solamente me funcionaban 3 pruebas y el tiempo de ejecución del resto resultaba que el proceso terminaba por lo que no podían ser resueltos.

Por otro lado, he ordenado el vector de valores antes de llamar a los algoritmos voraces para inicializar las cotas. Esto hace que encuentre la solución óptima más rápido por lo que el tiempo de ejecución es menor.

```
vector<int> v(valores.size());
for(size_t i = 0;i<v.size();i++) v[i] = i;

sort(v.begin(), v.end(), [&valores] (size_t x, size_t y){
    return valores[x] > valores[y];
});

vector<int> sorted_v(valores.size());
for(size_t j=0;j<valores.size();j++) sorted_v[j] = valores[v[j]];
```

5. Estudio comparativo de distintas estrategias de búsqueda.

```
javi@javi-VirtualBox:~/Escritorio/ADA/ADA_Final/bb_test_files$ ./maxsum-bb -f 1_bb.problem
98
60 59 32 29 28 0 0 1
0.016
javi@javi-VirtualBox:~/Escritorio/ADA/ADA_Final/bb_test_files$ ./maxsum-bb -f 10_bb.problem
325190892
40 39 14 27 27 0 0 0
0.014
javi@javi-VirtualBox:~/Escritorio/ADA/ADA_Final/bb_test_files$ ./maxsum-bb -f 20_bb.problem
713815015
2718 2717 1138 1581 1580 0 0 6
0.619
javi@javi-VirtualBox:~/Escritorio/ADA/ADA_Final/bb_test_files$ ./maxsum-bb -f 30_bb.problem
1056836703
11135694 11135693 5071503 6064192 6064175 0 0 16
2599.59
```

Como vemos los nodos completados en el algoritmo que he implementado no completa ningún nodo ya que se obtiene la solución óptima antes de llegar al último valor. Por lo que no obtendríamos la mejor solución a partir de los nodos completados tampoco. En el resto de ficheros ocurre lo mismo.

6. Tiempos de ejecución.

- Fichero 1_bb.problem: **0.015 ms.**
- Fichero 10_bb.problem: **0.014 ms.**
- Fichero 20_bb.problem: **0.625 ms.**
- Fichero 30_bb.problem: **2549.1 ms.**
- Fichero 40_bb.problem: **3077.54 ms.**
- Fichero 45_bb.problem: **2873.73 ms**.
- Fichero 50_bb.problem: **0.058 ms**.
- Fichero 100_bb.problem: **0.635 ms**.
- Fichero 200_bb.problem: **0.571 ms.**
- Fichero 1k_bb.problem: **6.297 ms.**
- Fichero 2k_bb.problem: **34.778 ms.**
- Fichero 3k_bb.problem: **73.283 ms.**