ANALISIS Y DISEÑO DE ALGORITMOS

Practica Final

Ramificación y poda

Grado en ingeniería informática

Francisco Joaquín Murcia Gómez 48734281H

Grupo 1

1) Estructuras de datos

1) Nodo

Nodo, está compuesto por la siguiente estructura: <int, int, Sol, int>, siendo Sol un vector de tipo int, que guarda las soluciones factibles, el primer int es la cota optimista, el segundo, es el valor que tiene ese nodo, y el ultimo int es el que controla para no recorrer los nodos recorridos.

2) Lista de nodos vivos

Para listar los nodos vivos usamos la librería "queue" para usar una cola de prioridad. Miramos primero si es factible, si el nuevo valor esta por debajo de la suma máxima, y después si ese valor esta por debajo o igual que la cota optimista.

En el código se ha implementado de la siguiente manera:

```
for (unsigned j = 0; j < 2; j++ ) { // non base
    x[k] = j;

    int new_value = value + x[k] * v[k]; // updating value

    if( new_value <= T ) {
        int valor_mejor=best_val;
        best_val = max( best_val, new_value + pesimista( v,  k+1, T - new_value));
        if(best_val!=valor_mejor)ACTUALIZADO_PESIMISTA++;//si el maximo no es el me
        int opt_bound = new_value + optimista( v,  k+1, T - new_value);
        if( opt_bound > best_val) {// is promising
            VIVOS++;
            pq.emplace( opt_bound, new_value,  x, k+1 );
        }else{
            DESCARTADOS_PRO++;
        }
    }else{
        DESCARTADOs_FAC++;
    }
}
```

2) Mecanismos de poda

1) Poda de nodos no factibles

Comprobamos si el valor nuevo es mayor que la suma máxima, en ese caso se descarta, como podemos observar:

```
for (unsigned j = 0; j < 2; j++ ) { // non base
    x[k] = j;

    int new_value = value + x[k] * v[k]; // updating value

    if( new_value <= T ) {
        int valor_mejor=best_val;
        best_val = max( best_val, new_value + pesimista( v,  k+1, T - new_value));
        if(best_val!=valor_mejor)ACTUALIZADO_PESIMISTA++;//si el maximo no es el m
        int opt_bound = new_value + optimista( v,  k+1, T - new_value);
        if( opt_bound > best_val) {// is promising
            VIVOS++;
            pq.emplace( opt_bound, new_value,  x, k+1 );
        }else{
            DESCARTADOS_PRO++;
        }
    }else{
        DESCARTADOs_FAC++;
    }
}
```

2) Poda de nodos no Prometedores

Si no es un nodo prometedor, es decir, en el "else" del "if" que controla los nodos vivos, como se puede ver a continuación:

```
for (unsigned j = 0; j < 2; j++ ) { // non base
    x[k] = j;

    int new_value = value + x[k] * v[k]; // updating value

    if( new_value <= T ) {
        int valor_mejor=best_val;
        best_val = max( best_val, new_value + pesimista( v,  k+1, T - new_value));
        if(best_val!=valor_mejor)ACTUALIZADO_PESIMISTA++;//si el maximo no es el me
        int opt_bound = new_value + optimista( v,  k+1, T - new_value);
        if( opt_bound > best_val) {// is promising
            VIVOS++;
            pq.emplace( opt_bound, new_value,  x, k+1 );
        }else{
            DESCARTADOS_PRO++;
        }
    }else{
        DESCARTADOs_FAC++;
    }
}
```

3) Cotas pesimistas y optimistas

1) Cota pesimista inicial.

Usamos el algoritmo voraz para la mochila discreta (sin los pesos), de este modo tomara un valor cercano.

El algoritmo es el siguiente:

```
int pesimista(  vector<int> &v,int k, int T) {


    int acc_v = 0;

    for( unsigned i=k;i<v.size();i++ ) {

        if( v[i] < T ) {
            acc_v += v[i];
            T -= v[i];
        }
    }

    return acc_v;
}
```

2) Cota pesimista del resto de nodos.

Para el resto de los nodos hacemos lo siguiente:

```
if( new_value <= T ) {
    int valor_mejor=best_val;
    best_val = max( best_val, new_value + pesimista( v,  k+1, T - new_value));
    if(best_val!=valor_mejor)ACTUALIZADO_PESIMISTA++;//si el maximo no es el me
    int opt_bound = new_value + optimista( v,  k+1, T - new_value);
    if( opt_bound > best_val) {// is promising
```

Hacemos la llamada con los siguientes valores, el vector de valores, añadiendo 1 a la variable que controla para no recorrer los nodos recorridos, y por último, el valor que falta para llegar a la suma máxima.

3) Cota optimista.

La cota optimista esta implementada a partir del siguiente algoritmo voraz:

```
int optimista(  vector<int> &v,int k, int T ){
    int acc_v = 0;
    for( unsigned i=k;i<v.size();i++) {
        if( v[i] > T ) {
            acc_v +=T;
            break;
        }
        acc_v += v[i];
        T -= v[i];
    }
    return acc_v;
}
```

4) Otros medios empleados para acelerar la búsqueda

Para acelerar la búsqueda, se ha implementado una poda,

```
if(value>=best_val)PROMETEDORES++;//sup

if(T==value) continue;//poda

if(value<=optimista( v,  k+1, T - value
```

De este modo controlamos si el valor es el máximo dejamos de buscar.

También hemos ordenado el vector de valores antes de iniciar el algoritmo.

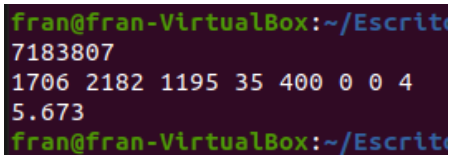
```
vector<int>vSorted(v.size());
for(unsigned i=0;i<v.size();i++){
    vSorted[i]=i;
}
sort(vSorted.begin(),vSorted.end(),[&v] (size_t x,size_t y){
    return v[x]>v[y];
});
vector<int>valores(vSorted.size());
for(unsigned i=0;i<vSorted.size();i++){
    valores[i]=v[vSorted[i]];
}

start = clock();
int k=podador(valores,T );
end=clock();
```

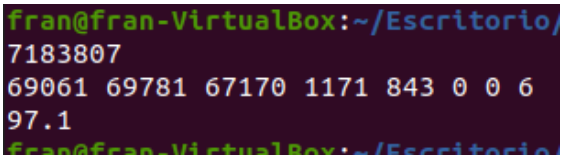
5) Estudio comparativo de distintas estrategias de búsqueda

Estas pruebas se han realizado con el fichero "Fichero 1k_bb.problem"

Usando todas las mejoras:



Sin las mejoras:



Como resultado tenemos que con mejoras es 17,34 veces más rápido.

Como se puede también observar, no aparecen nodos completados, esto es debido a que encontramos la solución antes de llegar a las hojas.

6) Tiempos de ejecución

FICHERO X_BB.PROBLEM	TIEMPO EN MS
1	0.015
10	0.012
20	0.698
30	2773.72
40	3409.78
45	3130.84
50	0.047
100	0.547
200	0.45
1K	6.231
2K	23.491
3K	52.677