Algoritmos para exploración en grafos

Algorítmica

Pablo M. Moreno Mancebo

Descripción del trabajo - Ejercicio 9

Dado un conjunto de n enteros, necesitamos decidir si puede ser descompuesto en dos subconjuntos disjuntos cuyos elementos sumen la misma cantidad. Resolver el problema mediante una técnica de exploración en grafos.

Índice

Análisis **3**

Diseño **3**

Componentes del algoritmo Backtracking 3

Adaptación de plantilla 4

Implementación **4**

1. Análisis

Creo que lo más inteligente es plantearlo de la siguiente forma, devolver un un vector de int en el cual, cada posición representa uno de los enteros, y el valor de esa posición puede ser 1 ó 2 (el conjunto al que lo metes).
He optado por **Backtracking:**

- Búsqueda exhaustiva para todas las posibles combinaciones que pueden ser solución

Si tenemos : 5 -3 2 0
Con Backtracking debería explorar todas las opciones posibles como:
Si en un subconjunto tenemos **5 + -3** = 2 y **2 + 0** = 2

2. Diseño

- Componentes del algoritmo Backtracking

- Buscar una **representación** del tipo T=(x1 , x2 , ..., xt): sol (vector de int en el cual, cada posición representa uno de los enteros, y el valor de esa posición puede ser 1 ó 2 (el conjunto al que lo metes).
- **Restricciones implícitas**: xi tendrá valores entre 1 y 2 dependiendo del conjunto donde se incluya
- **Restricciones explícitas:** Los subconjuntos deben ser disjuntos, no puede estar el mismo elemento en ambos subconjuntos.
- **Estructura del árbol/grafo implícito:** Los estados pueden ser:
  - 0 -> No se ha elegido conjunto
  - 1 -> Se ha metido en el conjunto 1
  - 2 -> Se ha metido en el conjunto 2
  - 3 -> Se ha probado tanto en el conjunto 1 como en el el 2
- **Función objetivo**:

```
SonIguales(secuencia, sol)
{
  s1 = 0, s2 = 0
  Desde  i = 0 hasta i < secuencia.size
    Si  sol[i] == 1  ENTONCES s1 += secuencia[i] SI NO  s2 += secuencia[i]
  Devolver (s1 == s2)
}
```

- Adaptación de plantilla

Para cada valor v posible de la variable $\mathbf{x_k}$ **hacer:**
Si se han probado todas las opciones, realizamos la vuelta atrás
SI NO
SI estamos en la última posición, se trata de una posible solución
SI hemos alcanzado el objetivo, devolvemos la solución,
SI NO, continuamos probando con el siguiente valor
SI NO
Probamos con el siguiente valor
FIN PARA
Devolvemos conjunto vacío por no encontrar solución.

3. Implementación

Compilar y ejecutar: **g++ -O2 -o ejecutable p5.cpp; ./ejecutable**

```
const int SIN_DECIDIR = 0;
const int TODOS_PROBADOS = 3;

//Función objetivo
bool SonIguales(const vector<int> &secuencia, const vector<int> &sol)
{
  int s1 = 0, s2 = 0;
  for (int i = 0; i < secuencia.size(); i++)
  {
     (sol[i] == 1) ? s1 += secuencia[i] : s2 += secuencia[i];
  }
  return (s1 == s2);
}

vector<int> Backtracking(const vector<int> &secuencia)
{
  // En el vector sol[i] =
  //  0 -> No se ha elegido conjunto
  //  1 -> Se ha metido en el conjunto 1
  //  2 -> Se ha metido en el conjunto 2
  //  3 -> Se ha probado tanto en el conjunto 1 como en el el 2
```

```
vector<int> sol(secuencia.size(), SIN_DECIDIR);
  int pos = 0;
  while (pos >= 0)
  {
     sol[pos]++;
     if (sol[pos] == TODOS_PROBADOS)
     {
 // Si se han probado todas las opciones, realizamos la vuelta atrás
        sol[pos] = SIN_DECIDIR;
        pos--;
     }
     else
     {
 // Si estamos en la última posición, se trata de una posible solución
        if (pos == secuencia.size() - 1)
        {
        // Si hemos alcanzado el objetivo, devolvemos la solución,
           // sino, continuamos probando con el siguiente valor
           if (SonIguales(secuencia, sol))
           {
              return sol;
           }
        }
        else
        {
       // Si no es solución, pasamos a probar con el siguiente valor
           pos++;
        }
     }
  }
  // No hemos encontrado solución, vector vacío
  return vector<int>();
}
```

4. Análisis teórico de la eficiencia

```
while (pos >= 0)O(2^n)
  {
     sol[pos]++;  O(1)
     if (sol[pos] == TODOS_PROBADOS)O(1)
     {
          sol[pos] = SIN_DECIDIR;O(1)
          pos--;O(1)
     }
     else
     {
        if (pos == secuencia.size() - 1)O(1)
           if (SonIguales(secuencia, sol))O(n)
                return sol;O(1)
        else
           pos++;O(1)

     }
  }
```

Explicación:

En el mejor caso, backtracking podría ser O(n), aunque podría llegar a tener que explorar el árbol completo del espacio de estados del problema.

En el peor caso, si el número de nodos es 2n , el tiempo del algoritmo backtracking será de orden O(p(n)2n) u O(q(n)n!), con p y q polinomios en n.

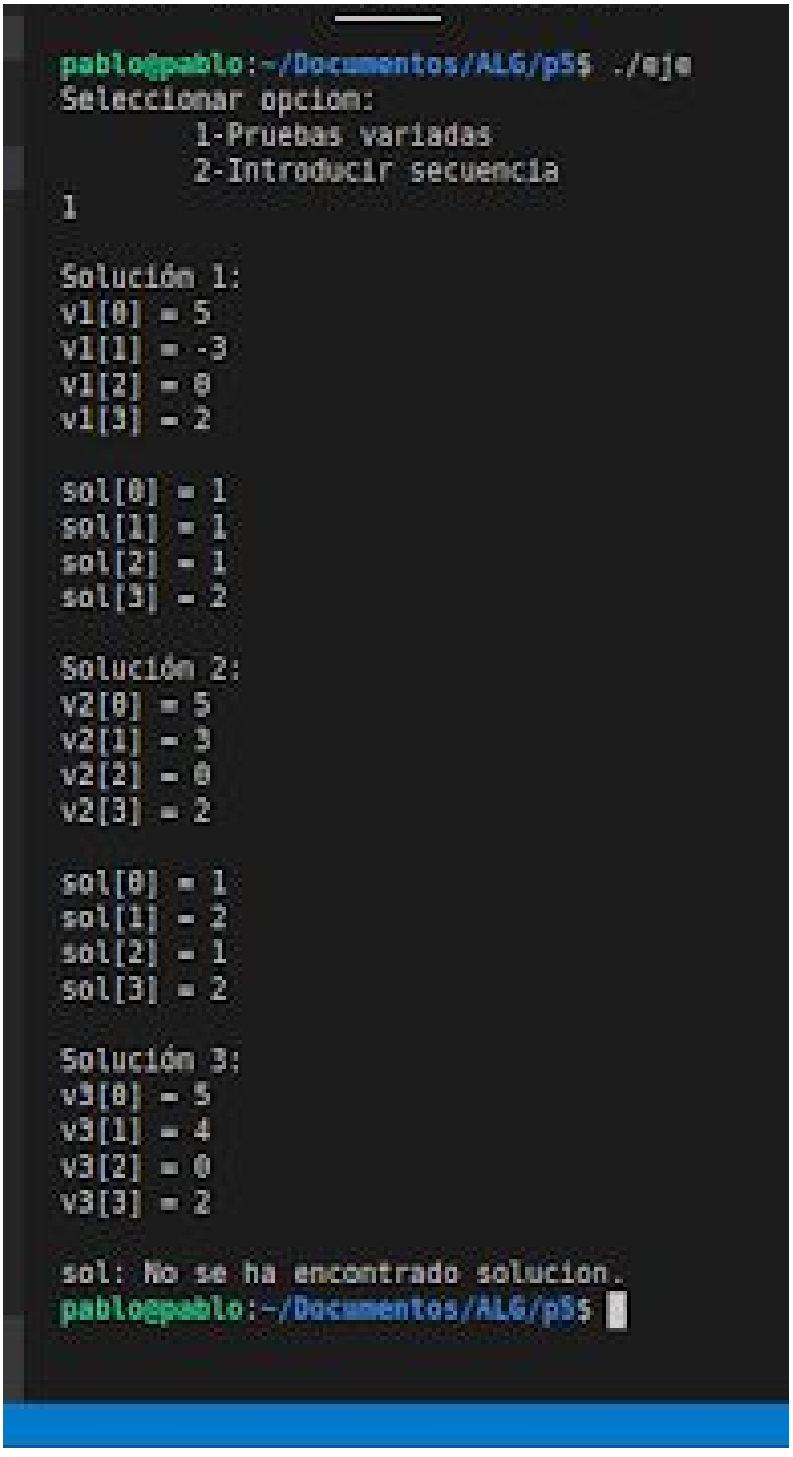
5. Pruebas: Ejecución del algoritmo

Compilar y ejecutar: **g++ -O2 -o ejecutable p5.cpp; ./ejecutable**

El main muestra un mini menú:

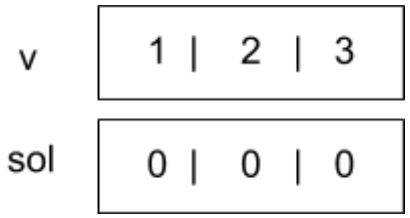
1- Hace 3 pruebas con números que he probado durante la realización del algoritmo

2- Pide los números para probar el algoritmo y muestra si la hay la posible organización de los subconjuntos.

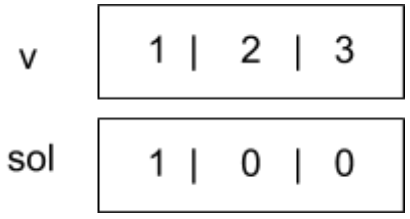


Ejecución paso a paso:

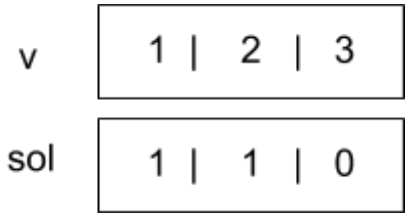
1º



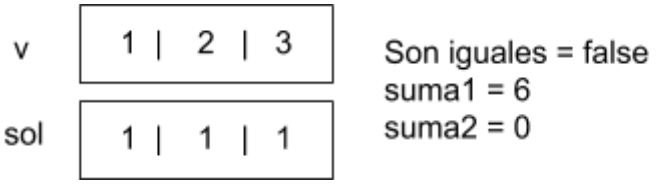
2º



3º



4º



5º