Ampliación de la Programación

Backtracking

Asignación de Trabajos

Asignación de Trabajos

Enunciado

N trabajos han de ser realizados por *N* personas, y se sabe que la persona *i* tardará un tiempo C_{ij}, en hacer el trabajo *j*.

Resuelva los siguientes problemas utilizando **Backtracking**:

a) Cada trabajo debe hacerlo una persona distinta y se debe minimizar el tiempo total.

b) Como en el apartado a) pero sin la restricción de que cada persona deba hacer un trabajo.

c) Como en a) pero minimizando el máximo tiempo para concluir los trabajos (todos los trabajadores comienzan simultáneamente).

Asignación de Trabajos

Resolución caso a)

Suponemos que los trabajos están identificados por 0, 1, …, N-1

En cada etapa k asignamos el trabajo k a una de las personas, que no tenga asignado ningún trabajo y supuesto que ya tenemos asignados los k-1 trabajos anteriores.

- **Test Solución**: Terminamos de construir una solución la etapa *N* (cuando hayamos asignado todos los trabajos).
  - **Comprobaremos** entonces si hemos o no mejorado nuestra mejor solución.
- **Test Fracaso**: No se puede asignar a una persona dos trabajos.
  - Esto lo comprobaremos con un vector de booleanos **Libres**

  Libre[i] = cierto si y solo si la persona i no tiene asignado aún algún trabajo
- **Generación de Descendientes**: El trabajo *k* lo puede hacer cualquiera de las personas (con el test de fracaso comprobamos que una persona no haya dos trabajos.

  Bastará para ello con un bucle:

  for (i=0; i<N; i++) // i persona que hace el trabajo

Asignación de Trabajos

Resolución caso a)

```
#include <stdio.h>
#define DIMMAX 20
#define INFINITO 10000
#define TRUE 1
#define FALSE 0

typedef struct {
   int coste;
   int persona [DIMMAX]; // persona[i] es la persona que realiza el trabajo i
 } TipoSolucion;

void AsignacionA (int k, int Libres[DIMMAX], TipoSolucion *sol, TipoSolucion *solOptima,
                  int N, int tiempo[DIMMAX][DIMMAX]);

void AsignacionB (int k, TipoSolucion *sol, TipoSolucion *solOptima,
                  int N, int tiempo[DIMMAX][DIMMAX]);

void AsignacionC (int k, int Libres[DIMMAX], TipoSolucion *sol, TipoSolucion *solOptima,
                  int N, int tiempo[DIMMAX][DIMMAX]);
```

Asignación de Trabajos

Resolución caso a)

```
void AsignacionA (int k, int Libres[DIMMAX], TipoSolucion *sol, TipoSolucion *solOptima,
                  int N, int tiempo[DIMMAX][DIMMAX]) {
  int p;

  if (sol->coste < solOptima->coste) // Vamos bien. Si NO FRACASO
    if (k == N)
      *solOptima = *sol;
    else
      for (p=0; p<N; p++)
        if (Libres[p]) {
          // Si la persona está libre lo puede hacer la persona p
          sol->persona[k] = p; sol->coste += tiempo[p][k];
          Libres[p] = FALSE;
            AsignacionA (k+1, Libres, sol, solOptima, N, tiempo);
          // deshacemos para la siguiente iteración
          sol->coste -= tiempo[p][k];
          Libres[p] = TRUE;
        }
}
```

Asignación de Trabajos

Resolución caso b)

```
void AsignacionB (int k, TipoSolucion *sol, TipoSolucion *solOptima,
                  int N, int tiempo[DIMMAX][DIMMAX]) {
    int p;

    if (sol->coste < solOptima->coste) // Vamos bien. Si NO FRACASO
      if (k == N)
        *solOptima = *sol;
      else
        for (p=0; p<N; p++) {
           sol->persona[k] = p; sol->coste += tiempo[p][k];
           AsignacionB (k+1, sol, solOptima, N, tiempo);

           // deshacemos para la siguiente iteración
           sol->coste -= tiempo[p][k];
         }
}
```

Asignación de Trabajos

Resolución caso c)

```
void AsignacionC (int k, int Libres[DIMMAX], TipoSolucion *sol, TipoSolucion *solOptima,
                  int N, int tiempo[DIMMAX][DIMMAX]) {
  int p, aux;

  if (sol->coste < solOptima->coste) // Vamos bien. Si NO FRACASO
    if (k == N)
      *solOptima = *sol;
    else
      for (p=0; p<N; p++)
        if (Libres[p]) {
          // Si la persona está libre lo puede hacer la persona p
          Libres[p] = FALSE;
          aux = sol->coste;
          sol->persona[k] = p; sol->coste = max(sol->coste, tiempo[p][k]);
          AsignacionA (k+1, Libres, sol, solOptima, N, tiempo);
          // deshacemos para la siguiente iteración
          sol->coste = aux;
          Libres[p] = TRUE;
        }
}
```