

1. (2 puntos) La ecuación en recurrencias resultante del análisis de eficiencia de un programa depende de una condición dada C , de modo que $T(n) = 2T(n/2) + n$ si C es cierta, y $T(n) = T(n/4) + n^2$ si C es falsa. En base a esta información, determine los órdenes \mathcal{O} y Ω del algoritmo. Justifique si el algoritmo tiene o no tiene orden Θ .

Para ello, bastará con resolver las recurrencias y sumar en cada caso /condición/

$$\cdot T(u) = 2T(u/2) + u \rightarrow u = 2^m \text{ luego } T(2^m) = 2T(2^{m-1}) + 2^m$$

• Parte homogénea

$$t_m - 2t_{m-1} = 0$$

$$\text{Luego } r_0 = 2, w_0 = 1$$

• Función de ajuste 2^m $2^m = 2^m \cdot 1^m$

$$\text{Luego } (x-2)^2 = 0 \text{ por tanto}$$

$$T(u) = c_0 2^m + c_1 u 2^m = c_0 u + c_1 u^2$$

$$\cdot T(u) = T(u/4) + u^2 \rightarrow u = 4^m \text{ luego } T(4^m) = T(4^{m-1}) + 4^m$$

Parte homogénea

$$t_m - t_{m-2} = 0$$

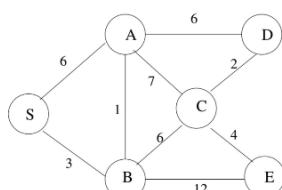
$$\begin{aligned} r_0 &= 1 & w_0 &= 1 \\ r_1 &= -1 & w_1 &= 1 \end{aligned}$$

Función de ajuste $4^m = 4^m \cdot 1^m$ luego

$$T(u) = c_0 + c_1 (-1)^m + c_2 4^m \Leftrightarrow T(u) = c_0 + c_1 (-1)^{\log_2 u} + c_2 u^2$$

Luego en cualquier caso $T(u) \in \mathcal{O}(u^2)$ y $T(u) \in \Omega(u)$ luego no tiene $\Theta(u)$

2. (2 puntos) Las galerías de una mina que conectan los puntos de extracción de mineral (A, B, C, D y E en la figura) con la salida (S en la figura) se han derrumbado. El coste de excavar de nuevo cada galería es conocido a priori (ver coste de los arcos del grafo en la figura). Se desea poder reabrir aquellas galerías que permitan conectar todos los puntos con la salida, pero con un coste lo menor posible. Diseñad un algoritmo eficiente que resuelva este problema en el caso general y aplicadlo a la instancia del ejemplo de la figura.

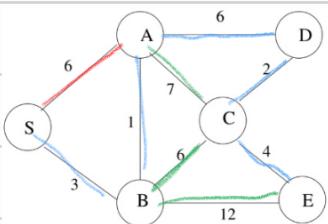
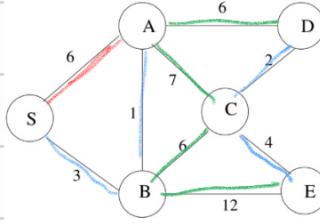
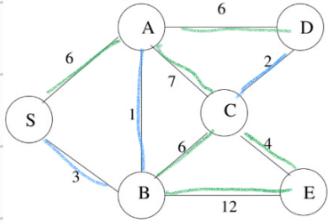
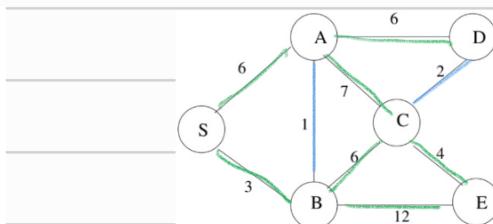
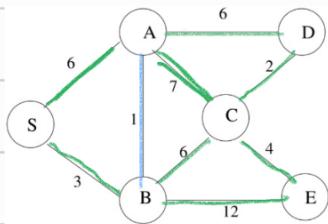


Este problema es un problema de árbol generador mínimo, específicamente de arcos positivos luego el algoritmo más eficiente posible, al no ser un grafo debo usar el algoritmo Kruskal.

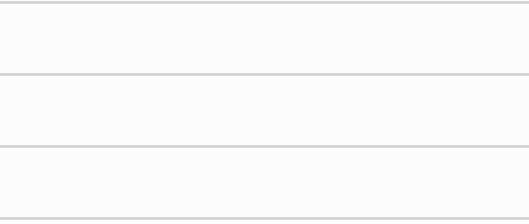
Por tanto, disponemos de los siguientes componentes:

- Cuadros. arcos del grafo
- Seleccionados arcos del grafo con menor peso que no forman ciclos
- Función de factibilidad: el conjunto de seleccionados no forma un ciclo
- Función objetivo: decidir si la solución alcanzada es una de todo el grafo
- Función solución: decidir si un conjunto de arcos une todos los puntos del grafo.
- Función selección: en cada iteración, se elige el arco con menor peso que no de lugar a ciclos en el conjunto de seleccionados

No es necesario probar la optimidad usando el código pues el usuario lo dirá así. En el ejemplo, se marcarán en azul los elegidos, en rojo los que no se eligieron por provocar ciclo y en verde todos los posibles cuadros a elegirse



FIN



4. (2 puntos) Un número natural N se dice que es cuadrado perfecto si se corresponde con el cuadrado de otro número natural. Por ejemplo, $4 = 2^2$, $25 = 5^2$, $100 = 10^2$ son cuadrados perfectos. Diseñe un algoritmo Divide y Vencerás que, dado un número natural N de entrada, determine si el número es cuadrado perfecto o no.

→ Algo parecido
a bisectiva

Asumiremos que , el uno y el cero son cuadrados perfectos

bool CuadradoPerfecto (int u) {
 if (u==0 || u==1) return true;
 else {

5. (2 puntos) Necesitamos realizar n tareas en una máquina multiprocesador, donde m procesadores pueden trabajar en paralelo. Sea t_i el tiempo de ejecución de la i -ésima tarea ($1 \leq i \leq n$). El problema consiste en implementar un algoritmo que determine en qué procesador hay que ejecutar cada tarea, de forma que el tiempo para ejecutar todas las tareas sea mínimo. Diseñe un algoritmo de exploración en grafos que resuelva el problema.

Usaremos la técnica de Backtracking, para ver que sabemos resolverlo hasta con crear la clase Solución, pues usando el esquema visto en clase se dará el resultado correcto. Dentro de la clase solución aparecerán los datos.

La solución será un vector de m componentes donde $v[i]$ podría tener los valores $1, 2, \dots, m$, es decir, representa el procesador asignado a la tarea i .

se usará un vector global $t[n]$ donde cada componente representa el tiempo a ejecutar cada tarea.

class Solucion {

private:

vector<int> v; int global;
int numTareas;
int numProc;

bool dentroRango (int u); } ; return (K>0 && u<numTareas);

public:

Solucion (int u, int v)

{ if ($u \geq 0$) {

 uProcareas = u;

 v.resize(u);

 for (int i=0; i < uProcareas; ++i) {

 v[i] = t;

 } } else {

 uProcareas = 0;

}

int Dicision (int u) {

 if (DeutroRango(u))

 return v[u];

}

bool Facilite () {

 bool facilite = SumaTiempPar() < global,

 facilite = absal() < global;

}

void ProcesaSolucion () {

 if (SumaTiempPar() < global) {

 global = SumaTiempPar();

 std::cout << V;

}

}

Por tanto, bastaría con aplicar el esquema de función SumaTiempPar para realizar la suma del vector, si no que toma el máximo de la suma de los tiempos sobre un mismo procesador, es $O(n^2)$

void InicPoupon (int u) {

 if (DeutroRango(u))

 v[u] = 0;

}

void SigValorPoupon (int u) {

 if (DeutroRango(u))

 v[u] += t;

 bool TadorGenerador (int u) {

 if (DeutroRango(u))

 return v[u] > uProcareas - 1;

}

 void Vueltas (int u) {

 if (DeutroRango(u))

 v[u] = 0;

}

 int size () {

 return uProcareas;

}

3. (2 puntos) Dado un tablero t de tamaño $n \times m$ de números naturales, se pretende resolver el problema de obtener el camino de la casilla $(0, 0)$ a la casilla $(n-1, m-1)$ que minimice la suma de los valores $t(i, j)$ de las casillas por las que pasa. En cada casilla (i, j) habrá sólo dos posibles movimientos: ir hacia abajo, a la casilla $(i+1, j)$, o ir hacia la derecha, a la casilla $(i, j+1)$ (siempre que no nos salgamos del tablero).

Diseñad un algoritmo lo más eficiente posible para resolver este problema utilizando programación dinámica. Aplicadlo para el siguiente tablero de tamaño 4×4 :

2	8	3	4
5	3	4	5
1	2	2	1
3	4	6	5

Vemos que es un problema de Programación Dinámica, para ello veamos que se cumple el P03. Esto es claro pues el mejor camino desde una casilla a a b es el mejor camino de una casilla intermedia a la final y de la inicial a la intermedia.

Unimos clavar por $d(i, j)$ al coste del camino de ir desde la casilla $(0, 0)$ hasta la casilla $d(i, j)$. Por tanto, sabemos que:

$$d(i, j) = \min \{ d(i-1, j) + c(i, j), d(i, j-1) + c(i, j) \}$$

$$d(0, 0) = c(0, 0)$$

$$d(i, 0) = \sum_{u=0}^i c(u, 0)$$

$$d(0, j) = \sum_{u=0}^j c(0, u)$$

Por tanto, el algoritmo queda como sigue, el hecho de que solo lleva dos caminos posibles es lo que provoca que lleva sólo dos considerados en el algoritmo.

para cada i desde 0 hasta $n-1$

$$d(i, 0) = \sum_{u=0}^i c(u, 0)$$

para cada j desde 0 hasta $m-1$

$$d(0, j) = \sum_{u=0}^j c(0, u)$$

para cada i desde 1 hasta $n-1$

para cada j desde 1 hasta $m-1$

$$d(i, j) = \min (d(i-1, j), d(i, j-1)) + c(i, j)$$

Esto hay que pensarlo mejor
 Esta bien pensado pensara vez bajando
 puedo subir luego es el único posible
 Al igual, una vez que desplaza a la derecha
 ya no me puedo desplazar a la izquierda

Aplicando el algoritmo al ejemplo:

$c_{i,j}$:

2 8 3 4
5 3 4 5
1 2 2 1
3 4 6 5

	0	1	2	3
0	2	10	13	17
1	7	10	14	19
2	8	10	12	13
3	11	14	18	18

Reseña, el camino más barato
es el que cuesta 18.

Aplicando el ejemplo, me he dado cuenta de que teniendo un greedy, obtendremos
de la solución óptima