UNIVERSIDAD DE GRANADA

E.T.S. DE INGENIERÍAS INFORMÁTICA y DE TELECOMUNICACIÓN

Departamento de Ciencias de la Computación e Inteligencia Artificial

Algorítmica

Guión de Prácticas

Práctica 3: Algoritmos Voraces (Greedy)

Curso 2021-2022

Grado en Informática

Objetivo

El objetivo de esta práctica es que el estudiante aprecie la utilidad de los métodos voraces (greedy) para resolver problemas de forma muy eficiente, en algunos casos obteniendo soluciones óptimas y en otros aproximaciones. Para ello cada equipo de estudiantes deberá resolver uno de los problemas (asignado al azar) que se describen en la sección 1. En todos los casos se debe especificar la formulación del problema dentro del esquema general de los algoritmos voraces, y también demostrar la optimalidad del algoritmo (o al menos dar argumentos o indicios sólidos que sugieran esa optimalidad). Adicionalmente, todos los equipos deberán diseñar e implementar algoritmos voraces (como heurística) para resolver el problema descrito en la sección 2.

1. Problemas

1.1. Salas de conferencias

Un centro educativo va a realizar un ciclo de n conferencias durante un día. Cada conferencia tiene establecido su horario, la conferencia i empieza a la hora s_i y termina a la hora f_i. Se desea que esta actividad interfiera lo mínimo posible con las actividades normales del centro. Por tanto, se quiere diseñar un algoritmo voraz que permita planificar todas las conferencias en su horario establecido usando el menor número de aulas posible (obviamente, dos conferencias no se pueden planificar en la misma aula si sus horarios se solapan). Demostrad la optimalidad del algoritmo.

1.2. Mínimo de los máximos de la suma de parejas de números

Se tienen n números naturales, siendo n una cantidad par, que tienen que juntarse formando parejas de dos números cada una. A continuación, de cada pareja se obtiene la suma de sus dos componentes, y de todos estos resultados se toma el máximo. Diseñad un algoritmo voraz que cree las parejas de manera que el valor máximo de las sumas de los números de cada pareja sea lo más pequeño posible, demostrando que la función de selección de candidatos usada proporciona una solución óptima.

Por ejemplo, supongamos que los datos se encuentran en el vector siguiente $[5, 8, 1, 4, 7, 9]$. Si seleccionamos las parejas (5, 8), (1, 4) y (7, 9), al sumar las componentes tenemos los valores 13, 5 y 16, por lo que el resultado final es 16. Si seleccionamos las parejas (5, 9), (8, 7) y (1, 4) al sumar las componentes tenemos 14, 15 y 5, por lo que el resultado final es 15 (mejor que antes). Si las parejas seleccionadas fuesen (1, 9), (4, 8) y (5, 7), las sumas son 10, 12 y 12, y el máximo es 12 (mejor aun).

1.3. Mínimo número de monedas

Se dispone de un conjunto de n monedas $\{m_1, m_2, \ldots, m_n\}$. El valor de la moneda m_i es igual a c_i, $i = 1, \ldots, n$, y el valor de todas las monedas es $\sum_{i=1}^{n} c_i = T$. Se quiere determinar el mínimo número de monedas (y cuáles son) necesario para cubrir al menos la cantidad M, es

decir que la suma de los valores de las monedas seleccionadas sea mayor o igual que M (donde $M \leq T$). Diseñad un algoritmo voraz para resolver este problema y demostrad su optimalidad.

1.4. Juntando vectores ordenados

Se tiene un conjunto de n vectores ordenados (de menor a mayor), $\{v_1, v_2, \ldots, v_v\}$. Cada vector v_i contiene t_i elementos, $i = 1, \ldots, n$. Se pretende irlos mezclando de dos en dos hasta conseguir un único vector también ordenado. La secuencia en la que se realiza la mezcla determinará la eficiencia del proceso, medida en términos del número total de asignaciones/movimientos que hay que realizar.

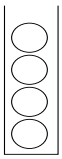
Por ejemplo, con $n = 3$ vectores de tamaños $t_1 = 30$, $t_2 = 20$ y $t_3 = 10$,

- Si mezclamos primero v_1 con v_2 se realizan 30+20=50 asignaciones, y el resultado se mezcla con v_3, realizándose entonces 50+10=60 asignaciones, con lo que se realizan en total 110 asignaciones.
- En cambio si mezclamos primero v_3 con v_2 se realizan 10+20=30 asignaciones, y el resultado se mezcla con v_1, realizándose ahora 30+30=60 asignaciones, con lo que se realizan en total 90 asignaciones.

Diseñad un algoritmo que determine la mejor forma de mezclar los vectores (el orden en que hay que mezclarlos) para minimizar el total de asignaciones. Demostrad que el algoritmo encuentra la solución óptima.

1.5. Guardando objetos

Se tienen n objetos esféricos $\{o_1, o_2, \ldots, o_n\}$ del mismo tamaño pero distintos pesos. Cada objeto o_i tiene un peso s_i, $i = 1, \ldots, n$. Esos objetos se van a almacenar en un recipiente en forma de tubo, de manera que para sacar un objeto hay que sacar primero todos los objetos anteriores (como en una pila, ver la figura). Se sabe con qué frecuencia se va a necesitar sacar cada uno de los objetos: el objeto o_i se necesita con probabilidad p_i (y por tanto $\sum_{i=1}^{n} p_i = 1$). Una vez usado el objeto, se guarda de nuevo en la misma posición.



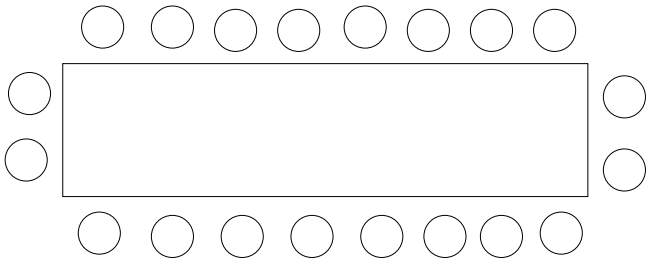
Si los objetos se guardan en orden $i_1, i_2, \ldots, i_n$ (o sea el primer objeto que se mete es o_{i_n}, el segundo objeto es $o_{i_{n-1}}$... y el último objeto es o_{i_1}), el peso total que hay que mover para extraer el objeto i_j es $\sum_{k=1}^{j} s_{i_k}$ (hay que mover todos los anteriores), y el peso total medio que hay que mover para extraer algún objeto (teniendo en cuenta la frecuencia con que se necesitan) es

$$\hat{P} = \sum_{j=1}^{n} \left[p_{i_j} \sum_{k=1}^{j} s_{i_k} \right]$$

Se desea minimizar el peso total medio empleando un algoritmo voraz que determine en qué orden almacenar los objetos. Demostrar lo siguiente, o dar un contraejemplo: podemos almacenar los objetos: (a) por orden no decreciente de s_i; (b) por orden no creciente de p_i; (c) por orden no creciente de p_i/s_i.

2. Conferencia de presidentes

Se va a celebrar una conferencia de presidentes a la que asistirán n personas. Todas se van a sentar alrededor de una única gran mesa rectangular, como la de la figura, de forma que cada persona tendrá sentadas junto a él a otras dos personas (una a su izquierda y otra a su derecha). En función de las características de cada persona (por ejemplo categoría o puesto, lugar de procedencia,...) existen unas normas de protocolo que indican el grado de conveniencia de que dos personas se sienten en lugares contiguos (supondremos que dicho grado es un número entero entre 0 y 100). El grado de conveniencia total de una asignación de personas a su puesto en la mesa es la suma de todos los grados de conveniencia de cada persona con cada una de los dos personas sentadas a su lado. Se desea sentar a las personas de forma que el grado de conveniencia global sea lo mayor posible.



Los asientos de la mesa los numeraremos como 0, 1, 2, hasta $n-1$, entendiendo que números contiguos representan asientos contiguos y que las posiciones 0 y $n-1$ también son contiguas. También representaremos a cada persona como un número entre 0 y $n-1$. Una instancia del problema será una matriz c, de tamaño $n \times n$, de enteros entre 0 y 100, donde $c[i][j]$ contiene el grado de conveniencia de sentar juntas a las personas i y j. No es necesario que la matriz sea simétrica, aunque lo supondremos por simplicidad. Una posible solución del problema se modelizará como un vector de enteros, a, de tamaño n, donde $a[i]$ es la persona que es asignada al asiento i-ésimo. Obsérvese que cada posible solución del problema es una permutación de los enteros de 0 a $n-1$ (si $i \neq j$ entonces $a[i] \neq a[j]$, una misma persona no se puede sentar en dos asientos distintos).

Para generar instancias del problema aleatoriamente se puede usar el siguiente generador:

```
double uniforme()
{
 int t = rand();
 double f = ((double)RAND_MAX+1.0);
 return (double)t/f;
}
```

```
void generaconveniencias() {
 srand(time(0));
 for (int i = 0; i < n-1; i++)
   for (int j = i+1; j < n; j++) {
        double u=uniforme();
        c[i][j]=(int)(u*101);
        c[j][i]=c[i][j];
     }
 for (int i = 0; i < n; i++) c[i][i]=0;
}
```

Para calcular el grado de conveniencia total de una asignación, usad este código:

```
int calculaconveniencia() {

int suma=0;
for (int i=1; i<n-1; i++)
   suma+=c[a[i]][a[i-1]]+c[a[i]][a[i+1]]; // desde 1 hasta n-2
suma+=c[a[0]][a[n-1]]+c[a[0]][a[1]]; //se añade los que están al lado de 0
suma+=c[a[n-1]][a[n-2]]+c[a[n-1]][a[0]]; //se añade los que están al lado de n-1
return suma;
}
```

El programa debe proporcionar como salida la asignación de personas a los asientos, así como el grado de conveniencia total de dicha asignación.

Para evaluar si los algoritmos voraces que se propongan se acercan más o menos a la solución óptima del problema, se pueden comparar sus resultados con la verdadera solución óptima, obtenida mediante una evaluación exhaustiva de los grados de conveniencia total de todas las asignaciones/permutaciones (quedándonos con la asignación de mayor conveniencia). Para ello se puede utilizar (modificándolo ligeramente) el algoritmo de B.R. Heap, que obtiene todas las permutaciones posibles de un vector dado:

```
void generapermutaciones(int m, int * a) {
    if (m==1) {
        for (int i=0; i < n; i++) cout <<a[i]<<" ";
        //sustituir la escritura de la permutacion por la evaluacion de la misma
        //e ir guardando la que vaya dando mejor conveniencia
    }
    else
        for (int i= 0; i < m; i++) {
            generapermutaciones(m-1, a);
            if (m%2==0) swap(a[i],a[m-1]);
            else
                swap(a[0],a[m-1]);
            }
}
```

Para llamar a ese procedimiento, hacer:

```
for (int i=0; i<n; i++)
    a[i]=i;
generapermutaciones(n,a);
```

Obviamente estas pruebas solo se pueden hacer para tamaños muy pequeños de n, ya que evidentemente el orden de eficiencia de este procedimiento es $O(n!)$.

2.1. Tareas a realizar

- Diseñad e implementad al menos un algoritmo voraz para resolver el problema anterior. Haced un estudio de su eficiencia (teórico, empírico e híbrido). Para tamaños pequeños comparad los resultados obtenidos con los de la verdadera solución óptima para tener idea de la calidad del algoritmo. **Nota importante**: estudiar la aplicación de los algoritmos voraces a la resolución del problema del viajante de comercio puede ser de ayuda en este problema.

- Si se ha diseñado más de un algoritmo, realizad un estudio comparativo de las estrategias propuestas, tanto desde el punto de vista de la eficiencia como de la calidad de los resultados.

- Se debe entregar una memoria detallada con todas las tareas realizadas (de las dos partes de la práctica), así como el código de todos los programas desarrollados. El informe debe entregarse en formato pdf. En el informe deben aparecer los nombres de todos los miembros del equipo que hayan participado en la realización de la práctica, así como el porcentaje de participación de cada miembro del equipo.