Examen parcial de Esquemas Algorítmicos

3 de abril de 2004

1. Estructuras de datos **3,5 puntos**

Hemos de diseñar soluciones eficientes para una serie de problemas seleccionando (o combinando) las estructuras de datos más adecuadas.

Debes indicar con claridad qué estructura de datos (simple u obtenida como combinación de estructuras más sencillas) te permite dar solución a cada problema con la mayor eficiencia posible. Acompaña tu respuesta con un breve análisis de complejidad espacial para la estructura de datos y temporal para las operaciones que debemos efectuar sobre ella.

(Recurre a gráficos ilustrativos de las estructuras si consideras que pueden resultar de ayuda.)

a) Las personas que iniciaron la colonización de una isla provienen todas de diferentes familias. Conforme transcurre el tiempo, los habitantes de la isla se casan entre sí y tienen descendencia. En los juzgados de la isla se registran todas la bodas y nacimientos.

   Hay una relación de parentesco natural entre padres, hijos y hermanos. Cuando dos personas se casan, establecen un vínculo de parentesco político y pasan a formar parte de una misma familia. La nueva relación de parentesco hace que todos los integrantes de las respectivas familias de los cónyuges pasen a considerarse miembros de una sola familia. O sea, dos personas son familia si existe una «cadena» de personas que son familia natural o política dos a dos.

   La estructura de datos más adecuada para representar este problema es el MF-set. Nos interesa modificarla para poder atender de la manera más eficiente posible a las dos siguientes situaciones:

   - Las leyes de la isla permiten que un individuo sea expulsado de una familia. Cuando eso ocurre, dicha persona pasa a constituir una nueva familia de la que es el único miembro. Describe cómo se podría realizar esa operación sobre la estructura.
   - Las reglas de cortesía de la isla indican que cuando un individuo se encuentra con otro de diferente familia, lo primero que hace es presentarse mediante la fórmula "Encantado de conocerle, los m miembros de mi familia estaríamos orgullosos si nos honrara con su amistad", donde m es el número total de miembros de su familia. Indica cómo puede disponer lo más rapidamente posible de dicha información y, en el caso en que sea necesario, detalla cómo afectaría la solución propuesta a las demás operaciones básicas de la estructura.

b) Una peña de apostantes juega todas las semanas a la lotería. Entre todos han nombrado a un responsable de hacer el seguimiento del resultado del sorteo. El acuerdo al que han llegado es que cada socio de la peña compra un décimo de lotería y le comunica al responsable el número (es posible que varios socios compren distintos décimos del mismo número). El responsable comprueba semanalmente el resultado del sorteo. El acuerdo que tienen es que, si toca algún premio, la mitad de su importe se lo queda la persona o personas que han comprado los décimos y la mitad restante se reparte a partes iguales entre todos los socios.

   El responsable debe poder introducir los números comprados por los socios durante la semana, comprobar si alguno de los números premiados está en poder de la peña y obtener los nombres de todos los socios que tienen cada uno de los décimos premiados lo más rápidamente posible.

c) En la universidad, tras finalizar el periodo de matrícula, para cada titulación se crea una lista de espera única para poder cubrir las posibles bajas, en la que están todos los estudiantes que deseándolo no han podido acceder a la titulación. Cada vez que se produce una baja en una titulación se da paso al estudiante con mejor nota de los que están en la lista de espera de esa titulación. Asimismo, es habitual que en diferentes periodos se incorporen nuevos estudiantes a las listas de espera, provinientes de la selectividad de septiembre, de segundos ciclos de formación profesional o de otras universidades. En cualquier caso, podemos asegurar que nunca habrá más de n estudiantes en cada lista de espera.

   Deseamos gestionar las listas de espera de forma que la selección de los estudiantes con mejores puntuaciones cuando se produzcan bajas y la incorporación de nuevos estudiantes a ellas sea lo más eficiente posible.

d) En un taller de la ITV para camiones, éstos deben pasar por una serie de controles (deben pasar necesariamente por todos ellos, aunque no es preciso que lo hagan en un orden concreto). Pese a estar en un mismo edificio, el acceso y salida de cada control se realiza de forma independiente de los demás controles a través de un largo y estrecho corredor en el que se suelen formar colas y que hace que, una vez el camión se incorpora al mismo, únicamente pueda salir tras superar el control o, si es el último de la cola, haciendo marcha atrás.

   En la empresa encargada de gestionar ese taller se han dado cuenta de que hay muchos momentos en que algunos controles quedan vacios, mientras en otros hay varios camiones esperando.

   Quisieran, por un lado, poder informar a la entrada del edificio a los nuevos camiones que llegan qué control es el que está más despejado. Por otra parte, si un control queda libre, les gustaría poder avisar por radio a un camión de los que están al final de una de las colas de acceso a los otros controles, para que haga marcha atras y se incorpore al control libre. ¿A qué camión? A aquel que (no habiendo pasado aún por el control que queda libre) está en la cola más larga.

e) Desarrolla un nuevo método para la clase *IndexedMinHeap*, llamado *increase*, que reciba un elemento que ya está en el heap y una nueva puntuación para el mismo e incremente su valor en el min-heap. El método debe ejecutarse en $O(\lg n)$, siendo n la talla del heap. Tras su ejecución los distintos elementos del heap indexado deben mantener sus propiedades.

En el caso en que optes por una descripción verbal, deberás acompañar a la misma necesariamente de un ejemplo gráfico que incluya la estructura completa.

2. Divide y vencerás **3,25 puntos**

La siguiente función calcula mediante la técnica de divide y vencerás el número de veces que aparece el elemento x en una lista desordenada L:

```
def veces(x,L):
  if len(L)==0:
    return 0
  elif len(L)==1:
    return 1
  else:
    return veces(x,L[0:len(L)/2])+veces(x,L[len(L)/2:len(L)])
```

a) ¿Es correcto el algoritmo? Si consideras que lo es, demuestra su correción. Si consideras que no lo es, corrígelo para que, aplicando la técnica de divide y vencerás, funcione adecuadamente y demuestra su corrección.

b) Presenta un análisis de complejidad temporal y espacial, indicando los costes en el mejor y peor de los casos si los hubiera. En el análisis del coste temporal utiliza el método del desplegado. Demuestra por inducción que la expresión cerrada a la que llegas es correcta.

c) Obtén una versión recursiva del algoritmo que no haga uso de la operación de corte. Analiza su complejidad computacional.

d) ¿Puedes proponer algún algoritmo iterativo que resuelva el mismo problema sin utilizar la técnica de divide y vencerás y que presente como mucho el mismo coste asintótico? (No es necesario que lo implementes, tan sólo descríbelo).

3. Divide y vencerás **3,25 puntos**

Sea a un vector de enteros diferentes (negativos y/o positivos) dispuestos en orden creciente ($a[0] < a[1] < \ldots < a[n-1]$) y sea i un número entero. El siguiente algoritmo responde a la pregunta de si el elemento que ocupa la posición i en la lista es precisamente i ($a[i] = i$):

```
def _buscar(i,a,p,r):
  if r-p==0:
    return False
  elif r-p==1:
    if i==a[p]:
      return True
    else:
      return False
  else:
    q=(r+p)/2
    if i<a[q]:
      return _buscar(i,a,p,q)
    else:
      return _buscar(i,a,q,r)
def buscar(i,a):
  return _buscar(i,a,0,len(a))
```

a) ¿Es correcto el algoritmo? Si consideras que no lo es, corrígelo para que, aplicando la técnica de divide y vencerás, funcione adecuadamente.

b) Presenta un análisis de complejidad temporal y espacial, indicando los costes en el mejor y peor de los casos si los hubiera.

c) Razona si hay o no hay recursión por cola en la versión original del algoritmo. En cualquier caso, elimina la recursión y obtén una versión iterativa. Analiza a continuación su complejidad espacial y temporal.

d) ¿Puedes proponer una versión del algoritmo original que mejore su comportamiento para el mejor caso? Sí es así, hazlo y analiza su complejidad computacional, indicando para que instancias del problema se comporta mejor el algoritmo.

e) Modifica el algoritmo original para que reciba un valor u que representa un umbral. Cuando se efectue una llamada al algoritmo en la que el subvector que se vaya a analizar tenga un tamaño menor o igual que dicho umbral, el algoritmo deberá sustituir las llamadas recursivas por una búsqueda secuencial del elemento en el vector.

f) ¿Se te ocurre algún método alternativo que resuelva el mismo problema sin utilizar la técnica de divide y vencerás y que iguale o mejore el coste asintótico del algoritmo original? (No es necesario que lo implementes, tan sólo descríbelo).

Examen parcial de Esquemas Algorítmicos

29 de mayo de 2004

1. Voraces — 3,5 puntos

Un alumno debe realizar a lo largo de un semestre N trabajos para otras tantas asignaturas. Cada trabajo le puede proporcionar hasta p_i puntos, para $1 \leq i \leq N$. El alumno conoce el tiempo máximo t_i, para $1 \leq i \leq N$, que tendría que invertir en cada trabajo para obtener la máxima puntuación. Si invierte menos tiempo, la puntuación del trabajo se verá reducida de forma proporcional (p.ej.: si en el trabajo i invierte $t_i/3$, la puntuación total será $p_i/3$). El tiempo máximo total que está dispuesto a invertir en todos los trabajos es T. ¿Cómo debe distribuir el tiempo T entre los trabajos de forma que obtenga la máxima cantidad global de puntos?

Se pide:

a) Formaliza el problema en términos de optimización,

b) propón una estrategia voraz que resuelva el problema y diseña un algoritmo que siga dicha estrategia devolviendo los tiempos que se deben dedicar a cada trabajo y

c) realiza un análisis de su complejidad computacional.

d) Suponiendo que al estudiante le exijan obtener una nota mínima en cada trabajo, $p'_i < p_i, 1 \leq i \leq N$, indica como afectaría esto a la formulación del problema y a la estrategia planteada para poder seguir resolviendo el problema de forma voraz. Señala si con esta restricción hay algún caso para el que no exista una solución factible para el problema.

2. Programación dinámica — 4,5 puntos

El problema de la asignación óptima de recursos puede resolverse mediante un algoritmo de programación dinámica. Recordemos su planteamiento inicial: disponemos de R unidades de un recurso y deseamos asignar cierta cantidad del mismo a cada una de D actividades distintas. Una función $v : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ nos indica el beneficio que obtenemos al asignar r unidades del recurso a la actividad d con $v(d, r)$. El número de unidades del recurso que podemos asignar a una actividad d debe ser inferior o igual a m_d. Deseamos obtener la asignación de recursos a actividades que maximiza el beneficio total obtenido, es decir, la suma de beneficios que proporciona cada asignación individual.

Considera la siguiente variante del problema: para cada actividad d no sólo hay un límite superior al número de unidades que se le pueden asignar (m_d), sino que también hay un límite inferior (un número mínimo de unidades de recurso) que se le deben asignar a la actividad (m'_d). Por otra parte, tenemos la obligación de consumir todas las unidades disponibles del recurso R.

a) Desarrolla un algoritmo de programación dinámica que determine cuál es la distribución de las unidades de recurso entre las actividades atendiendo a las condiciones de la variante planteada. Para ello:

   1) Formaliza el problema en términos de optimización,
   2) plantea la ecuación recursiva que calcule el valor de la solución óptima y
   3) diseña un algoritmo iterativo que devuelva dicha solución, estudiando su complejidad computacional.

b) ¿Es posible reducir la complejidad computacional si sólo estamos interesados en conocer el valor de la solución óptima? Razona la respuesta y, en caso afirmativo, di cuál es la complejidad espacial resultante.

c) Si sabemos que la máxima diferencia entre el límite inferior y el superior de las actividades que se pueden asignar a cualquier recurso es M (es decir, $M = \max_{1 \leq d \leq D}(m_d - m'_d)$), siendo $M < R$, ¿podrías expresar de forma más ajustada el coste temporal del algoritmo?

d) ¿Podría haber instancias del problema para las que no existiera una solución factible? Razona la respuesta y, si es así, indica las condiciones que se deberían cumplir para que pueda haber una solución factible y pon algún ejemplo de instancia que no tenga solución factible.

3. Búsqueda con retroceso — 2 puntos

En el problema de la suma de subconjuntos disponemos de N objetos con pesos $w_0, w_1, \ldots, w_{N-1}$ y de una mochila con capacidad para soportar una carga W y deseamos cargar la mochila con una selección arbitraria de objetos cuyo peso sea exactamente W. El siguiente algoritmo resuelve el problema aplicando la técnica de búsqueda con retroceso.

```
def subsets_sum(w, W):
  w.sort()
  return _subsets_sum(0, w, W, [0] * len(w))

def _subsets_sum(i, w, W, x):
  for x[i] in (1,0):
    total_weight = sum([w[j]*x[j] for j in range(i+1)])
    if total_weight == W:
      return x
    elif i < len(w)-1 and total_weight <= W:
      if i < len(w)-2 and total_weight + w[i+1] <= W:
        found = _subsets_sum(i+1, w, W, x)
        if found != None:
          return found
  return None
```

Dibuja el árbol de estados visitados por el algoritmo para una mochila con capacidad para $W = 15$ kilogramos y 5 objetos de pesos $w = [7, 3, 5, 2, 10]$. Al lado de cada estado, indica el valor del peso total ocupado por los objetos considerados. Señala con claridad cuál es la solución factible seleccionada y los pesos de los objetos que la forman.

Examen final de Esquemas Algorítmicos

29 de junio de 2004

1. Estructuras de datos — 1,25 puntos

Indica que estructura o estructuras de datos utilizarías para implementar algoritmos que resuelvan estos problemas y los costes temporales con los que se realizarían las correspondientes operaciones:

a) En un campamento con n niños se han formado m clanes cerrados, siendo m una cantidad desconocida. Cuando un niño quiere ingresar en un clan, debe solicitarlo a uno de sus miembros. Si éste accede, el niño es sometido a un ritual iniciático e ingresa en el clan. Cada clan tiene un cabecilla que es el fundador del mismo y al que no inició nadie. Un clan está formado por al menos dos personas. Deseamos acabar con esa práctica, para lo que necesitamos obtener un listado de clanes (de los nombres de los miembros de cada uno ellos, identificando al cabecilla), pero lo primero es conocer con detalle la situación actual.

   Para ello a cada niño se le pregunta si forma parte de un clan (puede responder sí o no) y, en caso de respuesta afirmativa, a qué persona solicitó su ingreso. Indica la estructura o estructuras de datos con que implementar eficientemente esta operación, el modo de identificar los cabecillas y cómo generar el listado. (Recuerda que un solo niño no forma un clan).

b) A los pacientes que llegan a urgencias en un hospital se les asigna un número entero de 0 a n atendiendo a la gravedad de su afección (n es la máxima gravedad y 0 es la mínima). Se pensó en usar un max-heap para implementar la lista de espera priorizada por gravedad, pero se observó una anomalía: cuando a dos o más pacientes se les asignaba una misma gravedad, podían «salir» del max-heap en cualquier orden. Es imperativo que, a igual gravedad, salgan en el mismo orden en que entraron, es decir, priorizados por el valor de la hora de entrada (que se conoce). Teniendo en cuenta esta restricción, ¿cómo podemos garantizar que el ingreso y la extracción de un paciente de una lista con m pacientes sean operaciones $O(\lg m)$?

2. Divide y vencerás — 2,75 puntos

Sea v un vector *ordenado* de elementos que pueden aparecer repetidos dispuestos en orden creciente ($v[0] \leq v[1] \leq \ldots \leq v[n-1]$). El siguiente algoritmo calcula mediante la técnica de divide y vencerás el número de veces que aparece repetido en el vector su *primer elemento*:

```
def veces(v,x):
    if len(v)==0:
        return 0
    elif len(v)==1:
        if x==v[0]:
            return 1
        else:
            return 0
    else:
        if x<v[len(v)/2]:
            if x==v[len(v)/2-1]:
                return len(v)/2
            else:
                return veces(v[:len(v)/2],x)
        else: #es decir, x==v[len(v)/2]
            return len(v)/2+veces(v[len(v)/2:],x)

print 'El elemento que ocupa la primera posición del vector aparece %d veces' % veces(v,v[0])
```

a) ¿Es correcto el algoritmo? Si consideras que lo es, demuestra su correción. Si consideras que no lo es, corrígelo para que, aplicando la técnica de divide y vencerás, funcione adecuadamente y demuestra su corrección.

b) Presenta un análisis de complejidad temporal y espacial, indicando los costes en el mejor y peor de los casos si los hubiera. En el análisis del coste temporal utiliza el método del desplegado.

c) Obtén una versión recursiva del algoritmo que no haga uso de la operación de corte. Analiza su complejidad computacional.

d) Propón un algoritmo iterativo que resuelva el mismo problema sin utilizar la técnica de divide y vencerás (no es necesario que lo implementes, tan sólo descríbelo). Indica el coste asintótico que presenta y si, atendiendo a dicho coste, puede considerarse mejor o peor que el anterior.

3. Voraces — 2 puntos

El problema de la selección de actividades puede resolverse eficientemente con una estrategia voraz. Te recuerdo su planteamiento básico: dada la propuesta de realizar en una sala una serie de n actividades, de las que conocemos la hora de comienzo y la de finalización, $C = \{(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)\}$, queremos seleccionar el mayor subconjunto de actividades que no se

solapen, ya que en la sala sólo se puede llevar a cabo una actividad en cada instante. Si una tarea acaba en un instante y otra empieza en ese mismo instante no hay conflicto alguno.

El algoritmo voraz se ejecuta en tiempo $O(n \lg n)$. ¿Es posible efectuar alguna modificación en cada uno de estos supuestos para obtener algoritmos más eficientes? Si es así, indica cómo y acompaña la explicación con el coste de los algoritmos resultantes.

a) Todas las actividades empiezan a la vez.

b) Ninguna de las actividades propuestas se solapa con las otras.

c) Todas las actividades duran lo mismo.

d) Las actividades se nos proporcionan ya ordenadas de menor a mayor instante de comienzo.

e) Las actividades se nos proporcionan ya ordenadas de menor a mayor instante de finalización.

f) Las actividades se nos proporcionan ya ordenadas de mayor a menor instante de finalización.

g) La hora de comienzo y de finalización de las actividades puede modificarse si ello facilita el encontrar una mejor combinación de las mismas, lo único que debería mantenerse es su duración (que se obtiene como $t_i - s_i, 1 \leq i \leq n$).

4. Programación dinámica — 2,75 puntos

Un vendedor ambulante de productos de artesanía realiza una ruta a través de una serie de P pueblos dispuestos a lo largo de una carretera. Parte de la ciudad en la que habita (situada al comienzo de la carretera) y debe llegar a una ciudad que se encuentra en el otro extremo de la carretera sin retroceder nunca. Su sistema de ventas es siempre el mismo: llega a un pueblo, vende toda la mercancía, compra la artesanía típica del pueblo y continúa hasta otro pueblo, en el que repetirá la operación. Al principio del viaje partirá con productos típicos de su ciudad y al acabar deberá vender la mercancía que le quede en la ciudad final de trayecto. El comerciante, una vez ha parado en un pueblo, no vuelve a detenerse hasta haber atravesado un mínimo de dos pueblos más (al ser pueblos vecinos no podría vender a buen precio los objetos).

El comerciante tiene una estimación del beneficio que puede obtener en un determinado pueblo en función de la mercancía que venda en él. Es decir, si numeramos los pueblos entre 1 y P según el orden en que aparecen en la carretera, el beneficio que obtiene al vender la mercancia de un pueblo i en otro j sería, $b(i, j)$, para $1 \leq i < j \leq P$.

Se pretende averiguar cual sería el máximo beneficio (en función del recorrido que haga) que podrá conseguir en su viaje. Se pide:

a) Formaliza el problema en términos de optimización,

b) plantea la ecuación recursiva que calcula el máximo beneficio,

c) representa esquemáticamente el grafo de dependencias entre las llamadas recursivas, calculando los costes espacial y temporal con los que se puede realizar el cálculo sobre dicho grafo (indica explicitamente justificando tu respuesta si es posible efectuar una reducción de complejidad espacial al efectuar el cálculo iterativo), y

d) diseña un algoritmo iterativo que recorra el anterior grafo y devuelva el *valor del máximo beneficio* que puede obtener el vendedor.

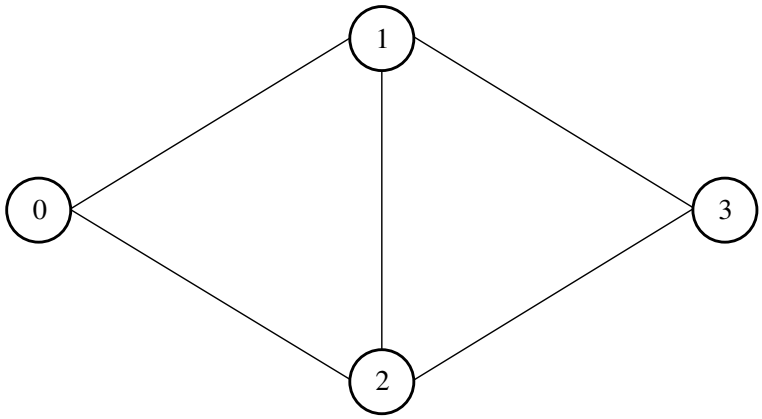
5. Búsqueda con retroceso — 1,25 puntos

En el problema de la existencia de un ciclo Hamiltoniano, dado un grafo $G = (V, E)$, deseamos encontrar un camino que parta de un vértice, termine en el mismo vértice y visite todos los vértices del grafo una sola vez (excepto en el caso del vértice de partida, que deberá coincidir con el de llegada). El siguiente algoritmo resuelve el problema aplicando la técnica de búsqueda con retroceso.

```
from sets import Set
def hamiltonian_cycle(G):
  return _hamiltonian_cycle(G, [G.V[0]]+[None]*(len(G.V)-1), 0, Set([G.V[0]]))

def _hamiltonian_cycle(G, path, m, visited_vertices):
  for v in G.succs(path[m]):
    if v not in visited_vertices:
      path[m+1] = v
      visited_vertices.add(v)
      if len(visited_vertices) == len(G.V):
        if (v,path[0]) in G.E:
          return path + [path[0]]
      else:
        found = _hamiltonian_cycle(G, path, m+1, visited_vertices)
        if found:
          return found
      visited_vertices.remove(v)
  return None
```

Dibuja el árbol de estados visitados por el algoritmo para el grafo mostrado a continuación (considera para ello que el vértice de partida es el 0 y que la función $G.succ(v), \forall v \in G.V$, devuelve *siempre* una lista con los vértices sucesores de v ordenados de menor a mayor). Incluye también, marcándolos de forma especial, aquellos estados que no llegan a visitarse por no cumplir la condición `if v not in visited_vertices`. Señala con claridad cuál es la solución factible seleccionada.



Examen segunda convocatoria de Esquemas Algorítmicos (IG24)

6 de septiembre de 2004

1. Estructuras de datos — 1,25 puntos

Indica qué estructura o estructuras de datos utilizarías para realizar operaciones que resuelvan estos problemas, el coste espacial de dichas estructuras y los costes temporales con los que se realizarían las operaciones indicadas:

a) En el parlamento valenciano los diputados autonómicos se adscriben obligatoriamente a un grupo parlamentario. Inicialmente los grupos se dan de alta y en la sesión inaugural de la legislatura cada diputado debe indicar, cuando es citado, el nombre del grupo parlamentario al que desea pertenecer. Una vez efectuada la adscripción inicial de todos los diputados querremos realizar dos tipos de operaciones: dado un grupo, obtener un listado de todos sus diputados, y dado un diputado, modificar su adscripción de un grupo a otro.

   Nos interesa poder realizar de la manera más eficiente la construcción de la estructura o estructuras que almacenen la información sobre diputados y grupos y las dos operaciones señaladas.

b) La UJI ha decidido crear la oficina de ayuda a la búsqueda de empleo (OFABE) de sus titulados recientes. Para ello, por cada titulación se mantendrá información sobre los alumnos egresados que lo soliciten, teniendo en cuenta tanto su nota media en la carrera como la antigüedad de la solicitud. Cuando llega una petición de personal desde una empresa, la OFABE avisa al titulado de la carrera requerida que presenta un mejor expediente (a igualdad de expediente, al de mayor antigüedad), eliminándolo de su lista tanto si acepta el empleo como si no lo hace (en este último caso, se avisaría al siguiente titulado, mientras que el titulado que rechazó el empleo no podrá solicitar de nuevo su inclusión en la lista hasta pasada una semana, perdiendo además la antigüedad acumulada). El procedimiento descrito es dinámico, ya que en cualquier momento deben poder añadirse nuevos demandantes de empleo (a los que se incorporará con los mismos derechos que los ya existentes, aunque con menor antigüedad), y la gente debe poder darse de baja voluntariamente de las listas de solicitantes.

   Nos interesa poder gestionar lo más eficientemente posible la información de las distintas titulaciones, en concreto la incorporación de nuevos titulados, la obtención de los candidatos a los empleos que se creen y las bajas voluntarias de las personas.

No olvides describir con claridad, cuando sea necesario, cómo las estructuras permiten modelar los problemas y, cuando plantees los costes, qué representan las variables con las que los indicas.

2. Divide y vencerás — 2,75 puntos

Sea v un vector de n elementos *distintos* que cumple que los elementos aparecen dispuestos en orden creciente desde el principio hasta una determinada posición p, y en orden decreciente desde esa posición p hasta el final del vector ($v[0] < v[1] < \ldots < v[p-1] < v[p] > v[p+1] > \ldots > v[n-2] > v[n-1]$), donde la posición p puede variar entre 0 y $n-1$ (es decir, puede estar en uno de los extremos del vector). El algoritmo `pos_mayor` calcula mediante la técnica de divide y vencerás la posición p que ocupa el elemento de mayor valor:

```
def pos_mayor(v):
    return _pos_mayor(v,0,len(v))

def _pos_mayor(v,i,k):
    if k-i==1:
        return i
    else:
        j=(i+k)/2
        if k-i>=3 and (v[j-1]<v[j] and v[j]>v[j+1]):
            return j
        else:
            if v[j-1]>v[j]:
                return _pos_mayor(v,j,k)
            else:
                return _pos_mayor(v,i,j)
```

a) ¿Es correcto el algoritmo? Si consideras que lo es, demuestra su correción. Si consideras que no lo es, corrígelo para que, aplicando la técnica de divide y vencerás, funcione adecuadamente y demuestra su corrección.

b) Realiza un análisis de complejidad temporal y espacial del algoritmo presentado, indicando los costes en el mejor y peor de los casos si los hubiera. En el análisis del coste temporal utiliza el método del desplegado.

c) Razona si hay o no hay recursión por cola en la versión original del algoritmo. En cualquier caso, elimina la recursión y obtén una versión iterativa. Analiza a continuación su complejidad espacial y temporal.

d) Propón un algoritmo iterativo que resuelva el mismo problema sin utilizar la técnica de divide y vencerás (no es necesario que lo implementes, tan sólo descríbelo). Indica el coste asintótico que presenta y si, atendiendo a dicho coste, puede considerarse mejor o peor que el anterior.

3. Voraces **2 puntos**

Con el fin de poder emitir certificados, la UJI dispone en su archivo de una fotocopia cotejada por cada uno de los títulos que ha expedido a sus alumnos desde su creación. Cada título tiene un número de registro único que lo identifica. En la actualidad, las fotocopias de los títulos se encuentran repartidas en varios montones, uno por cada titulación, y ordenadas, dentro de la titulación, en función de ese número. El nuevo archivero cree que resultaría más util disponer de una única clasificación para todas las copias de los títulos de la universidad, para lo cual ha decidido fusionar todos los montones en uno sólo, manteniendo la ordenación según el número de registro. Al estar ya ordenados dentro de cada titulación, ha decidido seguir el siguiente procedimiento: selecciona los montones de dos titulaciones y los fusiona tal y como haría el algoritmo `merge`, con lo que reduce en uno el número de montones; a continuación vuelve a hacer lo mismo con otro par de montones, y así sucesivamente hasta quedarse con ún único montón completamente ordenado.

Al archivero le gustaría poder *minimizar la cantidad total de comparaciones y fusiones entre pares de fotocopias* y cree que dicha cantidad puede depender tanto del número de fotocopias que hay cada montón (que se conoce) como del orden en que realice las fusiones. Por ello ha pensado en varias *estrategias voraces que le permitan determinar cómo seleccionar los montones que se deben fusionar en cada instante* (¡Ojo!, no te confundas, la estrategía voraz NO realiza las fusiones, sino sólo debe indicar cuál es el mejor orden para efectuarlas):

a) Fusionar en una primera etapa la titulación con mayor número de copias con la que tiene menos copias, la segunda con más copias con la segunda con menos copias y así sucesivamente. En una segunda etapa, seguir el mismo procedimiento con los montones resultantes. Repetir esta estrategia hasta quedarse con un único montón.

b) Fusionar en primer lugar las dos titulaciones que tienen un mayor número de copias. A continuación, fusionar el montón resultante con la siguiente titulación en (mayor) número de copías y aplicar este método sucesivamente hasta que quede solamente un montón.

c) Fusionar en primer lugar las dos titulaciones que tienen un menor número de copias. A continuación, fusionar el montón resultante con la siguiente titulación en (menor) número de copías y aplicar este método sucesivamente hasta que quede solamente un montón.

d) Seleccionar indistintamente los montones que se van a fusionar, ya que el número de comparaciones es independiente del orden en que se realizan las fusiones.

Indica para cada estrategia si encuentra o no la solución óptima. En este segundo caso, debes demostrarlo con un contraejemplo. Si crees que puede haber alguna otra estrategia correcta alternativa a las señaladas, descríbela. En todos los casos, indica y justifica el coste temporal de las estrategías voraces propuestas (el coste deberás expresarlo en función del número inicial de titulaciones/montones).

4. Programación dinámica **2,75 puntos**

El problema de la segmentación de un texto en palabras puede resolverse mediante un algoritmo de programación dinámica. Recordemos su planteamiento inicial: dada una secuencia t de caracteres sin espacios en blanco y dadas N palabras con sus probabilidades de aparición en el texto, deseamos segmentar la secuencia de caracteres en la secuencia de palabras más probable, para lo cual disponemos de una función `Pr` que nos indica, dada una secuencia de caracteres, la probabilidad de que dicha secuencia sea una palabra que aparezca en el texto. Las secuencias de caracteres que no pueden aparecer en el texto presentan probabilidad 0. La probabilidad de una secuencia de palabras se define como el producto de las probabilidades individuales de cada palabra.

Considera la siguiente variante del problema: conocemos el número exacto M de palabras que contiene el texto que se nos proporciona, por lo que cualquier segmentación que no contenga tal cantidad de palabras no podrá considerarse válida (¡aunque sea más probable!). Por otra parte, sabemos el tamaño W de la palabra más larga que puede aparecer en la frase.

Desarrolla un algoritmo de programación dinámica que determine cuál es la probabilidad de la mejor segmentación en M palabras del texto facilitado atendiendo a las condiciones de la variante planteada. Para ello:

a) Formaliza el problema en términos de optimización.

b) Plantea la ecuación recursiva que calcule la probabilidad de la segmentación óptima.

c) Diseña un algoritmo iterativo que devuelva el valor de la probabilidad de la segmentación óptima de la frase, estudiando su coste espacial y temporal.

d) Indica explícitamente, justificando tu respuesta, si es posible efectuar una reducción de la complejidad espacial del algoritmo. En tal caso, presenta una nueva versión del anterior algoritmo que incorpore esta mejora.

5. Búsqueda con retroceso **1,25 puntos**

En el problema de la suma de subconjuntos disponemos de N objetos con pesos $w_0, w_1, \ldots, w_{N-1}$ y de una mochila con capacidad para soportar una carga W y deseamos cargar la mochila con una selección arbitraria de objetos cuyo peso sea exactamente W. El algoritmo `subsets_sum` resuelve el problema aplicando la técnica de búsqueda con retroceso.

```
def subsets_sum(w, W):
  w.sort()
  return _subsets_sum(0, w, W, [0] * len(w))

def _subsets_sum(i, w, W, x):
  for x[i] in (1,0):
    total_weight = sum([w[j]*x[j] for j in range(i+1)])
    if total_weight == W:
      return x
    elif i < len(w)-1 and total_weight + w[i+1] <= W:
      found = _subsets_sum(i+1, w, W, x)
      if found != None:
        return found
  return None
```

Dibuja el árbol de estados visitados por el algoritmo para una mochila con capacidad para $W = 8$ kilogramos y 5 objetos de pesos $w = [1, 8, 5, 3, 6]$. Además,

a) Al lado de cada estado, indica de qué tipo es (prometedor, no prometedor, factible o no factible) y el valor del peso total ocupado por los objetos considerados.

b) Incluye también, marcándolos de forma especial, aquellos estados que no llegan a visitarse por no cumplir la condición `elif i <len(w)-1 and total_weight + w[i+1] <= W`.

c) Señala con claridad cuál es la solución factible seleccionada y los pesos de los objetos que la forman. Indica si para el conjunto de objetos propuestos hay otras soluciones factibles además de la devuelta por el algoritmo.