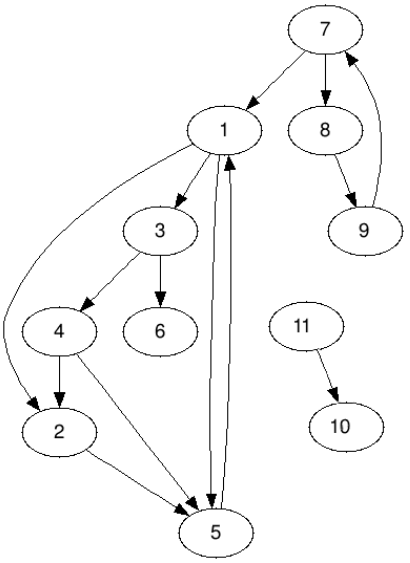
Examen Parcial de Programación 3 - TUDAI 14/6/2024



Ejercicio 1 *- No se pide implementar código en este ejercicio*

a) Dado el grafo G de la derecha se ejecuta el algoritmo de búsqueda en profundidad **DFS(G),** el cual visitará **todos** los vértices del grafo.
Suponiendo que el primer vértice que selecciona el algoritmo es el 1. Enumere, en el orden que va visitando el DFS: la lista de vértices que visita, la lista de arcos tree y **la lista de arcos back**.

b) Dado el grafo G de la derecha, **dar la lista de vértices** en el orden que se visita en un recorrido en amplitud **BFS(G)** del grafo suponiendo que el **primer vértice que selecciona el algoritmo es el 9.**
El BFS(G) finalizará cuando se hayan visitado **todos** los vértices del grafo.

Ejercicio 2

Se tiene un conjunto de N tareas que deben ser ejecutadas de manera secuencial (una detrás de la otra). Hay un único procesador. Se sabe que:

- Cada tarea tarda 1 segundo en ejecutar (todas tardan lo mismo 1 segundo).
- Cada tarea posee un puntaje asociado que se nos otorga luego de ejecutarla (la tarea i tiene un puntaje Pi).
- Cada tarea posee un valor de caducidad C que indica que se tiene que ejecutar entre las primeras C tareas (la tarea *i* tiene una caducidad Ci). *Por ej., si una tarea tiene C = 3, entonces para ser ejecutada debe estar dentro de las primeras 3 tareas a ejecutar sino ya no podrá ser ejecutada.*

Diseñe un algoritmo greedy que permita encontrar una secuencia S de tareas a ejecutar tal que se **maximice el puntaje** total obtenido. **Puede que NO todas las N tareas formen parte de S** debido a la restricción de caducidad.
*Por ejemplo, suponiendo las tareas T1 (P=10, C=2); T2 (P=20, C=1); T3 (P=8, C=2), la secuencia óptima S sería **[T2, T1]** sumando 30 puntos (20 puntos de T2, 10 puntos de T1). La tarea T3 no forma parte de la secuencia porque su caducidad C=2 no lo permite (ya que las primeras dos posiciones de la secuencia se encuentran ocupadas).*

a) ¿Explique cuál sería la estrategia **greedy** que seguiría?.
b) **Escriba un algoritmo en pseudo-java** que lo resuelva mediante la estrategia **greedy**.

Ejercicio 3

Dado un **conjunto C de letras** (que tiene más de 4 elementos), se desean generar **todas** las palabras válidas que contengan **exactamente 4 letras sin repetir y no empiecen con una letra vocal**. Se supone que contamos con una clase Diccionario que nos permite verificar si una secuencia de letras es una palabra válida `<Diccionario.esPalabraValida(string)>`.
Por ejemplo, con C={E, R, N, O, M, A, T} se debería generar una solución compuesta por {MANO, MONA, REMO, MORA, RAMO, ROEN, TOMA, ROTA, etc… }

a) Dibuje el **árbol de exploración** del algoritmo, indicando qué decisiones se toman en cada paso y qué información se lleva en los estados.
b) Escriba un **algoritmo en pseudo-java** que resuelva el problema mediante la técnica de **Backtracking**.
c) ¿Se podría plantear alguna **poda** que minimice la cantidad de estados generados por el Backtracking?

Ejercicio 4

a) **Muestre en JAVA** la **declaración de la clase Lista** simplemente vinculada, sus **atributos de instancia** y la **implementación** de sólo los siguientes dos métodos, **cada uno deberá tener complejidad O(1)**:
public void agregarPrincipio(<T> elemento)
public void agregarFinal(<T> elemento)
Suponga que la clase Nodo<T> ya está implementada con los métodos usuales.
Y responda (sin implementar) **¿qué complejidad tendría el método borrarUltimo?**, justifique su respuesta.
b) ¿Cuáles son las condiciones que deben darse para que la complejidad de la operación de búsqueda de un elemento en un árbol binario sea $O(\log_2 n)$?
c) Responda Verdadero o Falso, justificando su respuesta:
   i) En una estructura de hashing, definir un rho de diseño chico (por ejemplo 0,3) me asegura que no voy a tener listas de rebalse.
   ii) Luego de la ejecución del algoritmo de Dijkstra sobre un grafo G podemos decir que si en el array de distancias encontramos algún valor infinito, entonces el grafo G no es conexo.

Ejercicio 5

Dada la tabla de hashing de la derecha, que se comporta como el **HashTable de JAVA**, y se encuentra cargada de la manera que se muestra. Si **M=6 y ρ_d = 1**, **muestre cómo quedará la tabla si se inserta la clave 23**.