PRÁCTICA DE LABORATORIO Nro. 3

Carrera Computación

A. DATOS INFORMATIVOS		
Asignatura: Estructura de datos	**Ciclo / Semestre:** Tercero	**Paralelo:** A
Docente: José Oswaldo Guamán Quinche	**Período Académico:** Abril - septiembre 2023	
Integrantes: Cristian David Ajila Asanza		

B. INFORMACIÓN GENERAL	
Unidad: ANÁLISIS Y EVALUACIÓN DE ALGORITMOS.	
Tema: Algortimos de busqueda del camino mas corto	
Fecha: Loja, 30 de Mayo del 2023	**Nro. horas:** 2
Objetivos:	
• Realizar los ejercicios de evaluación de algoritmos	
Corresponde al resultado de aprendizaje:	
R1. Implementa algoritmos simples de búsqueda y explica las diferencias en el orden de complejidad computacional, bajo los principios de solidaridad, transparencia, responsabilidad y honestidad.	
Recursos y/o materiales:	
• Computador. • Material bibliográfico o recurso indicado en el EVA.	

C. DESARROLLO

Instrucciones:

Primera parte

Se requiere evaluar los siguiente algoritmos y sacar el orden de cada uno de ellos, estos están en la materia de evaluación de algoritmos diapositiva nro 24 a la 27. Ademas se requiere sacar el orden de magnitud de estos algortimos:

Algortimo 1

Initialize $Q(s, a)$ for all $s \in S, a \in A(s)$
for $i \leftarrow 1$ **to** *num_episodes* **do**
  $\epsilon \leftarrow$ setting new epsilon with ϵ-decay
  $\pi \leftarrow \epsilon$-greedy (Q)
  Select S_0
  Select A_0 from S_0 using π
  for $t \leftarrow 0$ **to** $T - 1$ **do**
    $R_{t+1}, S_{t+1} \leftarrow$ take action A_t from S_t in the environment
    $\pi \leftarrow \epsilon$-greedy (Q)
    Select A_{t+1} from S_{t+1} using π
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
    $S_t \leftarrow S_{t+1}$; $A_t \leftarrow A_{t+1}$
  end
end
return π

Algortimo 2

```
% % Datos
% f  = el nombre de la función como string
% a  = límite inferior
% b  = límite superior
% h = longitud del segmento
% n = numero de segmentos
% Resultados
% p=integración
  n=1;
  h = (b-a)/n;
  p1=h*(feval(f,a)+feval(f,b))/2;
  n=2;
  h = (b-a)/n;
  p2=h*(feval(f,a)+4*feval(f,(a+b)/2)+feval(f,b))/3;
  n=3;
  h=(b-a)/n;
  p3=3*h*(feval(f,a)+3*feval(f,a+h)+3*feval(f,a+2*h)+feval(f,b))/8;
  n=4;
  h=(b-a)/n;
  p4=4*h*(7*feval(f,a)+32*feval(f,a+h)+ ...
      12*feval(f,a+2*h)+32*feval(f,a+3*h)+7*feval(f,b))/90;
```

Algortimo 3

```
1  Program A-Estrela(G,w,inicio,alvo);
2  Begin
3       ls_fechada = ∅;
4       ls_aberta = inicio;
5       g[inicio] = 0;
6       f[inicio] = g[inicio] + h(inicio, alvo);
7       While ls_aberta ≠ ∅ do
8           u = ExtraiMin(ls_aberta);
9           ls_fechada = u;
10          For cada v ∈ Adj[u] do
11              if v ∈ ls_fechada then
12              else
13                  geraG = g[u]+w(u,v);
14                  if v ∉ ls_aberta or geraG < g[v] then
15                      π[v] = u;
16                      g[v] = geraG;
17                      f[v] = g[v] + h(v, alvo);
18                      if v == goal then
19                          return constroiCam(π[],v);
20                      if v ∉ ls_aberta then
21                          ls_aberta = v;
22      return failure;
23 End.
```

Algortimo 4

```
2     dist[source] ← 0                                  // Initialization
3     for each vertex v in Graph:
4         if v ≠ source
5             dist[v] ← infinity                                  // Unknown distance from source to v
6             prev[v] ← undefined                                  // Predecessor of v
7         end if
8         Q.add_with_priority(v, dist[v])
9     end for
10
11    while Q is not empty:                                 // The main loop
12        u ← Q.extract_min()                                 // Remove and return best vertex
13        for each neighbor v of u:
14            alt = dist[u] + length(u, v)
15            if alt < dist[v]
16                dist[v] ← alt
17                prev[v] ← u
18                Q.decrease_priority(v, alt)
19            end if
20        end for
21    end while
21    return dist[], prev[]
```

Nota: Todo los códigos deben guardarse en un repositorio git (gitlab, github o bitbucket)

Resolución:

Orden de Magnitud →
Algoritmo 1

Initialize $Q(s,a)$ for all $s \in S, a \in A(s)$	1
for $i \leftarrow 1$ **to** *num_episodes* **do**	n+1
$\epsilon \leftarrow$ setting new epsilon with ϵ-decay	n
$\pi \leftarrow$ ϵ-greedy (Q)	n
Select S_0	n
Select A_0 from S_0 using π	n
for $t \leftarrow 0$ **to** $T-1$ **do**	n
$R_{t+1}, S_{t+1} \leftarrow$ take action A_t from S_t in the environment	n*m+n
$\pi \leftarrow$ ϵ-greedy (Q)	n*m
Select A_{t+1} from S_{t+1} using π	n*m
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$	n*m
$S_t \leftarrow S_{t+1}$; $A_t \leftarrow A_{t+1}$	n*m
end	n*m
end	n
return π	1

Contador de frecuencia: 7n*m+7n+3
Orden: O(n^2)

Algoritmo 2

```
% % Datos                                                              
% f  = el nombre de la función como string                            
% a  = limite inferior                                                
% b  = limite superior                                                
% h = longitud del segmento                                           
% n = numero de segmentos                                             
% Resultados                                                          
% p=integración                                                       
  n=1;                                                                1
  h = (b-a)/n;                                                        n
  p1=h*(feval(f,a)+feval(f,b))/2;                                     n+1
  n=2;                                                                1
  h = (b-a)/n;                                                        1
  p2=h*(feval(f,a)+4*feval(f,(a+b)/2)+feval(f,b))/3;                  n+1
  n=3;                                                                1
  h=(b-a)/n;                                                          1
  p3=3*h*(feval(f,a)+3*feval(f,a+h)+3*feval(f,a+2*h)+feval(f,b))/8;   n+1
  n=4;                                                                1
  h=(b-a)/n;                                                          1
  p4=4*h*(7*feval(f,a)+32*feval(f,a+h)+ ...                           n+1
      12*feval(f,a+2*h)+32*feval(f,a+3*h)+7*feval(f,b))/90;           n+1
```

Contador de frecuencia: 6n+12
Orden: O(n)

Algoritmo 3

```
1  Program A-Estrela(G,w,inicio,alvo);                                
2  Begin                                                              
3      ls_fechada = ∅;                                                1
4      ls_aberta = inicio;                                            1
5      g[inicio] = 0;                                                 1
6      f[inicio] = g[inicio] + h(inicio, alvo);                       1
7      While ls_aberta ≠ ∅ do                                         n+1
8          u = ExtraiMin(ls_aberta);                                  n
9          ls_fechada = u;                                            n
10         For cada v ∈ Adj[u] do                                     m*n+1
11             if v ∈ ls_fechada then                                 
12             else                                                   
13                 geraG = g[u]+w(u,v);                               m*n
14                 if v ∉ ls_aberta or geraG < g[v] then              n*m
15                     M[v] = u;                                      m*n
16                     g[v] = geraG;                                  m*n
17                     f[v] = g[v] + h(v, alvo);                      m*n
18                     if v == goal then                              m*n
19                         return constroiCam(M[],v);                 m*n
20                     if v ∉ ls_aberta then                          m*n
21                         ls_aberta = v;                             m*n
22     return failure;                                                n
23 End.                                                               
```

Contador de frecuencia: 10m*n+4b+6

Algoritmo 4

```
2     dist[source] ← 0                                  1
3     for each vertex v in Graph:                       n
4         if v ≠ source                                 n
5             dist[v] ← infinity                        n
6             prev[v] ← undefined                       n
7         end if
8         Q.add_with_priority(v, dist[v])               n
9     end for
10
11    while Q is not empty:                             n+1
12        u ← Q.extract_min()                           n
13        for each neighbor v of u:                     n*m
14            alt = dist[u] + length(u, v)              n*m
15            if alt < dist[v]                          n*m
16                dist[v] ← alt                         n*m
17                prev[v] ← u                           n*m
18                Q.decrease_priority(v, alt)           n*m
19            end if                                    n*m
20        end for                                       n*m
21    end while                                         n*m
21    return dist[], prev[]                             1
```

Contador de frecuencia: 8n*m+7n+3

Ejercicios del libro—> Orden de Magnitud →

```
1.1   sub_programa  p1(n)                               1
          s = 0                                         1
          for  i = 1 to n  do                           n
               for  j = 1 to i  do                      n(n+1)/2+n
                    for  k = 1 to j  do                 n((n+1)/2+n)/2+n
                         s = s + 1                      n((n+1)/2+n)/2
                    end(for)                            n((n+1)/2+n)/2
               end(for)                                 n(n+1)/2
          end(for)                                      n
      fin(sub_programa)                                 1
```

Contador de frecuencia: (3n(n(n+1)/2+n)/2)+(2n(n+1)/2)+4+2
Orden: (n^3)

```
1.2   sub_programa  p2(n)
          s = 0                                         1
          i = 1                                         1
          j = 1                                         1
          while  i <= n  and  j <= n  do                (n+1)
               s = s + 1                                n
               i = i + 1                                n
               if  i > n  and  j < n  ther               (n+1)*n
                    i = 1                               n^2
                    j = j + 1                           n^2
               end(if)                                  n^2
          end(while)                                    n
          write(s)                                      1
      fin(sub_programa)
```

Contador de frecuencia: 4n^2+5n+n
Orden: O(n^2)

```
1.3   sub_programa  p3(vec,n,x)
          i = 1                              1
          j = n                              1
          do                                 n+1
                k = (i+j)/2                  n
                if  vec(k) <= x  then        m+1
                      i = k + 1              m
                else                         m
                      j = k - 1              m
                end(if)                      m
          while  i <= j                      1
      fin(sub_programa).
```

Contador de frecuencia: 4m+2n+4
Orden: O(1)

```
1.4   sub_programa  p4(vec,n)
          i = 1                              1
          while  i <= n  do                  n+1
                s = 0                        n
                j = i + 1                    n(n+1)*n
                while  j <= n  and  j <= i + vec   n*n
                      s = s + vec(i)         n*n
                      j = j + 1              n*n
                end(while)                   1
                write(s)                     1
                i = j                        n
          end(while)
      fin(sub_programa)
```

Contador de frecuencia: 4(n)^2+5n+4
Orden: O(n)^2

```
1.5   Sub_programa  p5()
            read(n)                          1
            s = 0                            1
            i = 2                            1
            while  i  <=  n   do             log2n+n
                  s = s + 1                  log2n
                  i = i * i                  log2n
            end(while)                       log2n
            write(n,s)                       1
        fin(sub_programa)
```

Contador de frecuencia: 4log2n+n+4
Orden: O(log2n)

```
1.6   Sub_programa p5()
          read(n)                          1
          s = 0                            1
          i = 3                            1
          while i <= n do                  log3n+n
              s = s + 1                    log3n
              i = i * i                    log3n
          end(while)                       log3n
          write(n,s)                       1
      fin(sub_programa)
```

Contador de frecuencia: 4log3n+n+4
Orden: O(log3n)

Link del video de Youtube
https://youtu.be/OBbr1CF6GhI
Link de github
https://github.com/CrisDaa7/Practicas_de_Tercero.git

Conclusiones:

-Al finalizar la practica podemos darnos cuenta de la importancia de la evaluación de los algoritmos, ya que estos nos ayudan a comprender su rendimiento y eficiencia.
-Podemos concluir que al analizar cada algoritmo podemos tener una idea de como escala su tiempo de ejecución o requerimientos de espacio a medida que aumenta el tamaño de la entrada.
-Finalmente podemos decir que siempre tratamos de buscar los algoritmos con órdenes de magnitud más bajos, ya que tienden a ser más eficientes.

D. RÚBRICA DE EVALUACIÓN

Nota: En caso de no cumplir con alguno de los parametros establecidos se calificara la nota igual a 0
Si se encuentra copia con algun compañero o prácticas realizadas de otros años, o bajados del internet, se aplicara el reglamento de deshonestidad estudiantil y se calificará sobre 0.
No se aceptará trabajos atrasados, se calificará sobre 0.
Todo acerca de deshonestidad academica que no diga este documento.

Informe de trabajo: • Contenido: pertinente y concreto; • Estructura y organización: Elementos vinculados y estructurados coherentemente. • Originalidad y creatividad: trabajo inédito, presentación de nuevas ideas.	2 ptos
Resolución de Ejercicios: • Proceso de resolución de ejercicios: con originalidad y creatividad usando lo aprendido en clases • Ejecución de programas y solución de la aplicación (debe estar todo solucionado al 100%)	6 ptos
Conclusiones: • Redacción • Originalidad y creatividad: conclusiones inéditas en base a su experiencia y objetivos planteados.	1 ptos
Video • Se debe entregar un video con la demostración y explicando el código de su solución.	1 ptos
Total **(Ponderado en calificación final 25%)**	10 ptos

E. FIRMAS DE RESPONSABILIDAD DE LO ACTUADO

Estudiante(s):	**Firma**

Cristian David Ajila Asanza