



UNIVERSIDAD AUTONOMA
DE CHIHUAHUA

ESTRUCTURA DE DATOS

ÁRBOLES Y GRAFOS.

- **DOCENTE:** PERLA IVONNE CORDERO DE LOS RIOS
- **ALUMNO** JONATHAN GANDARA SALAZAR
- **MATRICULA** 374357
- **FECHA** 20 02 2024



JohnTenno

All rights reserved © John Tenno

GRAFOS

GRAFO:

UN GRAFO ES UNA ESTRUCTURA COMPUESTA POR UN CONJUNTO DE NODOS (VÉRTICES) Y UN CONJUNTO DE ARISTAS (CONEXIONES ENTRE LOS NODOS).

TIPOS DE GRAFOS:

DIRIGIDOS: LAS ARISTAS TIENEN UNA DIRECCIÓN, INDICANDO UNA RELACIÓN DE UN NODO A OTRO.

NO DIRIGIDOS: LAS ARISTAS NO TIENEN DIRECCIÓN, INDICANDO UNA RELACIÓN BIDIRECCIONAL ENTRE LOS NODOS.

REPRESENTACIÓN DE GRAFOS:

MATRIZ DE ADYACENCIA: UNA MATRIZ CUADRADA DONDE LAS FILAS Y COLUMNAS REPRESENTAN NODOS Y LAS ENTRADAS INDICAN LA EXISTENCIA DE UNA ARISTA.

LISTA DE ADYACENCIA: UN ARRAY DE LISTAS DONDE CADA ELEMENTO DEL ARRAY REPRESENTA UN NODO Y CONTIENE UNA LISTA DE NODOS ADYACENTES.

PROPIEDADES:

CONECTIVIDAD: INDICA SI HAY UN CAMINO ENTRE CADA PAR DE NODOS EN EL GRAFO.

CICLICIDAD: INDICA SI EL GRAFO CONTIENE CICLOS.

PODERACIÓN: LOS GRAFOS PUEDEN SER PONDERADOS, DONDE LAS ARISTAS TIENEN UN PESO ASOCIADO.

ARBOLES

DEFINICIÓN DE ÁRBOL:

UN ÁRBOL ES UNA ESTRUCTURA DE DATOS JERÁRQUICA QUE CONSISTE EN NODOS, CON UN NODO RAÍZ Y NODOS HIJOS, FORMANDO UNA RELACIÓN DE ANCESTRO-DESCENDIENTE.

TIPOS DE ÁRBOLÉS:

BINARIOS:

CADA NODO TIENE A LO SUMO DOS HIJOS.

BINARIOS DE BÚSQUEDA:

DONDE EL VALOR DEL NODO IZQUIERDO ES MENOR Y EL VALOR DEL NODO DERECHO ES MAYOR QUE EL VALOR DEL NODO PADRE.

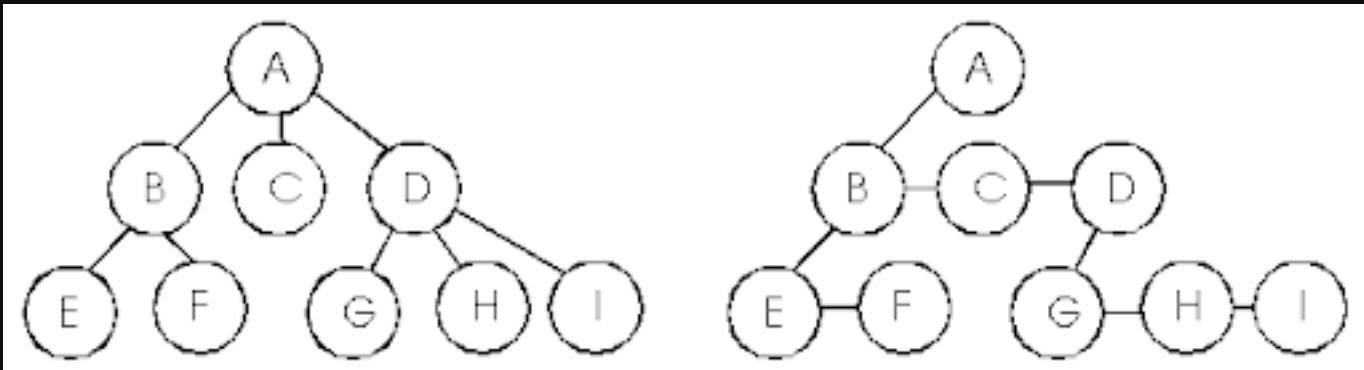
AVL:

ÁRBOL BINARIO DE BÚSQUEDA AUTO-BALANCEADO DONDE LAS ALTURAS DE LOS DOS SUBÁRBOLES DE UN NODO DIFIEREN EN NO MÁS DE UNO.

B-TREES:

ÁRBOL DE BÚSQUEDA AUTO-BALANCEADO DISEÑADO PARA SISTEMAS QUE LEEN Y ESCRIBEN GRANDES BLOQUES DE DATOS.

CONVERSIÓN DE ÁRBOL GENERAL A BINARIO



CONVERSIÓN DE CADA NODO:

EL PRIMER HIJO DE CADA NODO EN EL ÁRBOL GENERAL SE CONVIERTA EN EL HIJO IZQUIERDO EN EL ÁRBOL BINARIO.

EL HERMANO MÁS CERCANO DEL HIJO SE CONVIERTA EN EL HIJO DERECHO EN EL ÁRBOL BINARIO.
PROCEDIMIENTO RECURSIVO:

SE DEBE REPETIR DE MANERA RECURSIVA PARA CADA NODO Y SUS HIJOS.

DIFERENCIA ENTRE ÁRBOLES BINARIOS COMPLETOS Y CASI COMPLETOS

ÁRBOL BINARIO COMPLETO:

TODOS LOS NIVELES DEL ÁRBOL, EXCEPTO POSIBLEMENTE EL ÚLTIMO, ESTÁN COMPLETAMENTE LLENOS.

EL ÚLTIMO NIVEL ESTÁ LLENO DE IZQUIERDA A DERECHA SIN HUECOS.

ÁRBOL BINARIO CASI COMPLETO:

SIMILAR AL ÁRBOL BINARIO COMPLETO, PERO PERMITE QUE EL ÚLTIMO NIVEL NO ESTÉ COMPLETAMENTE LLENO.

LOS NODOS ESTÁN LO MÁS CERCA POSIBLE DEL NIVEL SUPERIOR.

EXPLICACIÓN DE ALGORITMOS

ÁRBOLES DE BÚSQUEDA BINARIOS (BST):

UN ÁRBOL BINARIO EN EL QUE CADA NODO TIENE UN VALOR MAYOR QUE TODOS LOS VALORES EN SU SUBÁRBOL IZQUIERDO Y MENOR QUE TODOS LOS VALORES EN SU SUBÁRBOL DERECHO.
OPERACIONES BÁSICAS: INSERCIÓN, BÚSQUEDA, Y ELIMINACIÓN.

ÁRBOLES ENHILADOS:

UTILIZAN PUNTEROS ADICIONALES PARA APUNTAR AL SUCESOR IN-ORDER Y PREDECESOR IN-ORDER, FACILITANDO EL RECORRIDO SIN RECURSIÓN.
PERMITEN REALIZAR UN RECORRIDO EN INORDEN DE MANERA EFICIENTE.

ÁRBOLES BALANCEADOS (AVL):

ÁRBOLES BINARIOS DE BÚSQUEDA AUTO-BALANCEADOS, DONDE LA DIFERENCIA DE ALTURA ENTRE SUBÁRBOLES DE UN NODO ES COMO MÁXIMO 1.
UTILIZAN ROTACIONES (SIMPLES Y DOBLES) PARA MANTENER EL BALANCE.

ÁRBOLES B:

ÁRBOLES DE BÚSQUEDA AUTO-BALANCEADOS QUE PERMITEN MÁS DE DOS HIJOS POR NODO, DISEÑADOS PARA FUNCIONAR BIEN EN SISTEMAS DE ALMACENAMIENTO EN DISCO.
MANTIENEN EL BALANCE MEDIANTE LA DIVISIÓN Y FUSIÓN DE NODOS.

```

● ● ●
1 #include <iostream>
2 #include <stdlib.h>
3 #include <queue>
4 #include <unordered_set>
5 #include <stack>
6
7 using namespace std;
8
9 struct nodo
10 {
11     char nombre; // nombre del vertice o nodo
12     struct nodo *sgte;
13     struct arista *ady; // puntero hacia la primera arista del nodo
14 };
15
16 struct arista
17 {
18     struct nodo *destino; // puntero al nodo de llegada
19     struct arista *sgte;
20     struct arista *ady;
21 };
22
23 typedef struct nodo *Tnodo; // Tipo Nodo
24 typedef struct arista *Tarista; // Tipo Arista
25
26 Tnodo p = NULL; // puntero cabeza
27
28 void menu();
29 void insertar_nodo();
30 void agrega_arista(Tnodo &, Tnodo &, Tarista &);
31 void insertar_arista();
32 void vaciar_aristas(Tnodo &);
33 void eliminar_nodo();
34 void eliminar_arista();
35 void mostrar_grafo();
36 void mostrar_aristas();
37 void BFS();
38 void BFSutil(Tnodo start);
39 void DFSutil(Tnodo start);
40
41 int main()
42 {
43     int op; // opción del menú
44
45     system("color 0b");
46
47     do
48     {
49         menu();
50         cin >> op;
51
52         switch (op)
53         {
54             case 1:
55                 insertar_nodo();
56                 break;
57             case 2:
58                 insertar_arista();
59                 break;
60             case 3:
61                 eliminar_nodo();
62                 break;
63             case 4:
64                 eliminar_arista();
65                 break;
66             case 5:
67                 mostrar_grafo();
68                 break;
69             case 6:
70                 mostrar_aristas();
71                 break;
72             case 7:
73                 BFS();
74                 break;
75             case 8:
76                 DFS();
77                 break;
78             case 9:
79                 break;
80             default:
81                 cout << "OPCIÓN NO VALIDA...!!!" << endl;
82                 break;
83         }
84
85     } while (op != 9);
86
87     cout << endl;
88     system("pause");
89     system("cls");
90
91 } while (op != 9);
92
93 return 0;
94 }
95
96 void menu()
97 {
98     cout << "\n\tREPRESENTACION DE GRAFOS DIRIGIDOS\n\n";
99     cout << " 1. INSERTAR UN NODO" << endl;
100    cout << " 2. INSERTAR UNA ARISTA" << endl;
101    cout << " 3. ELIMINAR UN NODO" << endl;
102    cout << " 4. ELIMINAR UNA ARISTA" << endl;
103    cout << " 5. MOSTRAR GRAFO" << endl;
104    cout << " 6. MOSTRAR ARISTAS DE UN NODO" << endl;
105    cout << " 7. RECORRIDO EN AMPLITUD (BFS)" << endl;
106    cout << " 8. RECORRIDO EN PROFUNDIDAD (DFS)" << endl;
107    cout << " 9. SALIR" << endl;
108
109    cout << "\n INGRESE OPCIÓN: ";
110 }
111
112 void insertar_nodo()
113 {
114     Tnodo t, nuevo = new struct nodo;
115     cout << "INGRESE VARIABLE: ";
116     cin >> nuevo->nombre;
117     nuevo->sgte = NULL;
118     nuevo->ady = NULL;
119
120     if (p == NULL)
121     {
122         p = nuevo;
123         cout << "PRIMER NODO...!!!" << endl;
124     }
125     else
126     {
127         t = p;
128         while (t->sgte != NULL)
129         {
130             t = t->sgte;
131         }
132         t->sgte = nuevo;
133         cout << "NODO INGRESADO...!!!" << endl;
134     }
135 }
136
137 void agrega_arista(Tnodo &aux, Tnodo &aux2, Tarista &nuevo)
138 {
139     Tarista q;
140     if (aux->ady == NULL)
141     {
142         aux->ady = nuevo;
143         nuevo->destino = aux2;
144         cout << "PRIMERA ARISTA....!" << endl;
145     }
146     else
147     {
148         q = aux->ady;
149         while (q->sgte != NULL)
150         {
151             q = q->sgte;
152         }
153         nuevo->destino = aux2;
154         q->sgte = nuevo;
155         cout << "ARISTA AGREGADA...!!!!" << endl;
156     }
157 }
158
159 void insertar_arista()
160 {
161     char ini, fin;
162     Tarista nuevo = new struct arista;
163     Tnodo aux, aux2;
164
165     if (p == NULL)
166     {
167         cout << "GRAFO VACIO...!!!!" << endl;
168         return;
169     }
170     nuevo->sgte = NULL;
171     cout << "INGRESE NODO DE INICIO: ";
172     cin >> ini;
173     cout << "INGRESE NODO FINAL: ";
174     cin >> fin;
175     aux = p;
176     aux2 = p;
177     while (aux2 != NULL)
178     {
179         if (aux2->nombre == fin)
180         {
181             break;
182         }
183         aux2 = aux2->sgte;
184     }
185     while (aux != NULL)
186     {
187         if (aux->nombre == ini)
188         {
189             agrega_arista(aux, aux2, nuevo);
190             return;
191         }
192         aux = aux->sgte;
193     }
194 }
195
196 void vaciar_aristas(Tnodo &aux)
197 {
198     Tarista q, r;
199     q = aux->ady;
200     while (q->sgte != NULL)
201     {
202         r = q;
203         q = q->sgte;
204         delete (r);
205     }
206 }
207
208 void eliminar_nodo()
209 {
210     char var;
211     Tnodo aux, ant;
212     aux = p;
213     cout << "ELIMINAR UN NODO" << endl;
214     if (p == NULL)
215     {
216         cout << "GRAFO VACIO...!!!!" << endl;
217         return;
218     }
219     cout << "INGRESE NOMBRE DE VARIABLE: ";
220     cin >> var;
221
222     while (aux != NULL)
223     {
224         if (aux->nombre == var)
225         {
226             if (aux->ady != NULL)
227                 vaciar_aristas(aux);
228
229             if (aux == p)
230             {
231                 p = p->sgte;
232                 delete (aux);
233                 cout << "NODO ELIMINADO...!!!!" << endl;
234                 return;
235             }
236             else
237             {
238                 ant->sgte = aux->sgte;
239                 delete (aux);
240                 cout << "NODO ELIMINADO...!!!!" << endl;
241             }
242         }
243     }
244 }
245
246 void eliminar_arista()
247 {
248     char ini, fin;
249     Tnodo aux, aux2;
250     Tarista q, r;
251     cout << "ELIMINAR ARISTA" << endl;
252     cout << "INGRESE NODO DE INICIO: ";
253     cin >> ini;
254     cout << "INGRESE NODO FINAL: ";
255     cin >> fin;
256     aux = p;
257     aux2 = p;
258     while (aux2 != NULL)
259     {
260         if (aux2->nombre == fin)
261         {
262             break;
263         }
264         aux2 = aux2->sgte;
265     }
266     while (aux != NULL)
267     {
268         if (aux->nombre == ini)
269         {
270             if (aux->ady == aux2)
271             {
272                 aux->ady = aux2->sgte;
273                 delete (aux2);
274                 cout << "ARISTA ELIMINADA....!!!!" << endl;
275             }
276             else
277             {
278                 aux = aux->sgte;
279             }
280         }
281     }
282 }
283
284 void mostrar_grafo()
285 {
286     Tnodo ptr;
287     Tarista ar;
288     ptr = p;
289     cout << "NODO|LISTA DE ADYACENCIA" << endl;
290
291     while (ptr != NULL)
292     {
293         cout << " " << ptr->nombre << "|";
294         if (ptr->ady != NULL)
295         {
296             ar = ptr->ady;
297             while (ar != NULL)
298             {
299                 cout << " " << ar->destino->nombre;
300                 ar = ar->sgte;
301             }
302         }
303         ptr = ptr->sgte;
304     }
305 }
306
307 void mostrar_aristas()
308 {
309     Tnodo aux;
310     Tarista ar;
311     char var;
312     cout << "MOSTRAR ARISTAS DE NODO" << endl;
313     cout << "INGRESE NODO: ";
314     cin >> var;
315     aux = p;
316     while (aux != NULL)
317     {
318         if (aux->nombre == var)
319         {
320             if (aux->ady == NULL)
321             {
322                 cout << "EL NODO NO TIENE ARISTAS...!!!!" << endl;
323                 return;
324             }
325             else
326             {
327                 cout << "NODO " << aux->nombre << " TIENE ARISTAS" << endl;
328                 ar = aux->ady;
329                 while (ar != NULL)
330                 {
331                     cout << " " << ar->destino->nombre;
332                     ar = ar->sgte;
333                 }
334             }
335         }
336     }
337 }
338
339 void BFS()
340 {
341     char start;
342     cout << "INGRESE EL NODO INICIAL PARA BFS: ";
343     cin >> start;
344
345     Tnodo aux = p;
346     while (aux != NULL && aux->nombre != start)
347     {
348         aux = aux->sgte;
349     }
350
351     if (aux != NULL)
352     {
353         BFSutil(aux);
354     }
355     else
356     {
357         cout << "NODO NO ENCONTRADO" << endl;
358     }
359 }
360
361 void BFSutil(Tnodo start)
362 {
363     if (start == NULL)
364     {
365         return;
366     }
367     unordered_set<char> visited;
368     queue<Tnodo> q;
369     visited.insert(start->nombre);
370     q.push(start);
371
372     while (!q.empty())
373     {
374         Tnodo current = q.front();
375         q.pop();
376         cout << current->nombre << " ";
377
378         Tarista ar = current->ady;
379         while (ar != NULL)
380         {
381             if (visited.find(ar->destino->nombre) == visited.end())
382             {
383                 visited.insert(ar->destino->nombre);
384                 q.push(ar->destino);
385             }
386             ar = ar->sgte;
387         }
388     }
389 }
390
391 void DFS()
392 {
393     char start;
394     cout << "INGRESE EL NODO INICIAL PARA DFS: ";
395     cin >> start;
396
397     Tnodo aux = p;
398     while (aux != NULL && aux->nombre != start)
399     {
400         aux = aux->sgte;
401     }
402
403     if (aux != NULL)
404     {
405         DFSutil(aux);
406     }
407     else
408     {
409         cout << "NODO NO ENCONTRADO" << endl;
410     }
411 }
412
413 void DFSutil(Tnodo start)
414 {
415     char start;
416     cout << "INGRESE EL NODO INICIAL PARA DFS: ";
417     cin >> start;
418
419     Tnodo aux = p;
420     while (aux != NULL && aux->nombre != start)
421     {
422         aux = aux->sgte;
423     }
424
425     if (aux != NULL)
426     {
427         DFSutil(aux);
428     }
429     else
430     {
431         cout << "NODO NO ENCONTRADO" << endl;
432     }
433 }
434
435 void DFSUtil(Tnodo start)
436 {
437     if (start == NULL)
438     {
439         return;
440     }
441     unordered_set<char> visited;
442     stack<Tnodo> s;
443     visited.insert(start->nombre);
444     s.push(start);
445
446     while (!s.empty())
447     {
448         Tnodo current = s.top();
449         s.pop();
450         cout << current->nombre << " ";
451
452         Tarista ar = current->ady;
453         while (ar != NULL)
454         {
455             if (visited.find(ar->destino->nombre) == visited.end())
456             {
457                 visited.insert(ar->destino->nombre);
458                 s.push(ar->destino);
459             }
460             ar = ar->sgte;
461         }
462     }
463 }
464
465 void BFSUtil(Tnodo start)
466 {
467     if (start == NULL)
468     {
469         return;
470     }
471     unordered_set<char> visited;
472     stack<Tnodo> s;
473     visited.insert(start->nombre);
474     s.push(start);
475
476     while (!s.empty())
477     {
478         Tnodo current = s.top();
479         s.pop();
480         cout << current->nombre << " ";
481
482         Tarista ar = current->ady;
483         while (ar != NULL)
484         {
485             if (visited.find(ar->destino->nombre) == visited.end())
486             {
487                 visited.insert(ar->destino->nombre);
488                 s.push(ar->destino);
489             }
490             ar = ar->sgte;
491         }
492     }
493 }

```