AED1 Practica T4

Ángel Ruiz Fernández G2.2
José Alberto García Berna

1. Listado de problemas resueltos

PID	Algoritmo	Represenación	Comodin	SID
402	BPA	Matrix de Ady.		1171
403	BPP	Lista de Ady.		1889
405	Simulado	Lista de Aris.	Si, pls	2072

*PID: Problema
*SID: Numero de envío

2. Resolución de problemas

2.1. 402

2.1.1. Analisis, diseño y eficiencia

El grafo se representa mediante una matriz de adyacencia, ya que es la mas simple. Además existe un array booleano que representa si cada nodo ha sido recorrido.

La busqueda primero en anchura se efectua mediante la funcion ‘wide_search‘. Se resetea el array de recorridos y se empieza un bucle, donde por cada nodo sin recorrer, se llama a una función ‘bpa‘.

‘bpa‘ contiene un algoritmo iterativo con una una cola. La cola se inicializa con el nodo de inicio (y se imprime), y mientras esa cola no esté vacía, se toma un nodo de la cola y se se itera por todos los nodos adyacentes a ese, que no hayan sido recorridos: cada uno se marca como recorrido, y se inserta a la cola, a la vez que se imprime.

La dificultad del ejercicio reside visualizar como y cuando se ha de insertar en la cola, para mi.

Ya que en este se usa una matriz de adyacencia, es O(n+a), siendo n el numero de nodos y a el numero de aristas, complejidad lineal.

2.1.2. Listado del código

402.cpp 75L 1345B

```
 1 #include <iostream>
 2 #include <queue>
 3
 4 constexpr int MAX_NOD = 26;
 5
 6 bool ma[MAX_NOD][MAX_NOD] = { };
 7 bool checked[MAX_NOD] = { };
 8
 9 void clear_ma() {
10     for (int y = 0; y < MAX_NOD; y++)
11         for (int x = 0; x < MAX_NOD; x++)
12             ma[x][y] = false;
13 }
14
15 void read_graph(int A) {
16     clear_ma();
17
18     char X, Y;
19     for (int j = 0; j < A; j++) {
20         std::cin >> X >> Y;
21         ma[X - 'A'][Y - 'A'] = true;
22     }
23 }
24
```

402.cpp 75L 1345B

```
25 bool exists(int v) {
26     for (int i = 0; i < MAX_NOD; i++)
27         if (ma[v][i] || ma[i][v]) return true;
28     return false;
29 }
30
31 void bpa(int N, int v) {
32     std::queue<int> C;
33     checked[v] = true;
34
35     std::cout << char(v + 'A');
36
37     C.push(v);
38
39     while (!C.empty()) {
40         int x = C.front();
41         C.pop();
42         for (int y = 0; y < N; y++) {
43             if (ma[x][y] && !checked[y]) {
44                 checked[y] = true;
45                 C.push(y);
46                 std::cout << char(y + 'A');
47             }
48         }
49     }
50 }
51
52 void wide_search(int N) {
53     for (int i = 0; i < MAX_NOD; i++)
54         checked[i] = false;
55
56     for (int i = 0; i < N; i++)
57         if (!checked[i])
58             ws(N, i);
59     std::cout << std::endl;
60 }
61
62 int main() {
63     int T, N, A;
64
65     std::cin >> T;
66
67     for (int i = 0; i < T; i++) {
68         std::cin >> N >> A;
69         read_graph(A);
70         wide_search(N);
71     }
72 }
73
74
75
```

2.2. 403

2.2.1. Analisis, diseño y eficiencia

El laberinto empieza en el primer nodo, y se sale por el ultimo nodo.

El grafo se representa mediante una lista de nodos, y de cada elemento cuelga una lista de nodos adyacentes; y un booleano de visitado.

La función ‘main‘ de entrada llama a la funcion ‘read_graph‘ que lee el grafo y retorna su representación. Esta es pasada a ‘search‘ que se encarga de llamar a ‘bpp‘, que contiene un algoritmo variante de la busqueda en profundidad, recursiva, que retorna bool, empezando por el primer nodo.

Empieza por marcar el nodo actual como visitado, y lo añade a un vector pasado por referencia que representa el camino tomado. Si el nodo actual es el nodo de salida, se retorna falso (detener); si no, se itera por todos los nodos adyacentes al actual que no hayan sido visitados, y se recursa por cada uno.

Si la llamada recursiva retorna falso, se retorna falso también, para acabar el arbol de recursión cuando se encuentra el nodo de salida, pero si retorna verdadero, se ha de seguir, insertando el nodo al vector de camino y continuando.

En este ejercicio me atasqué bastante, ya que tampoco caí cuando insertar al camino tomado en el bucle de bpp, llegando a crear estructuras de datos extra, de ayuda, hasta que me desatasqué imaginandolo un ejemplo en el papel.

Ya que es una variante de busqueda primero en profundidad, es de complejidad temporal O(n + a).

2.2.2. Listado del código

403.cpp 76L 1656B

```
 1 #include <iostream>
 2 #include <vector>
 3 #include <stack>
 4 #include <sstream>
 5 #include <utility>
 6
 7 constexpr int MAX_NOD = 26;
 8
 9 std::vector<std::pair<std::vector<int>, bool>> read_graph() {
10     int n, a;
11     std::string line;
12     auto al = std::vector<std::pair<std::vector<int>, bool>>();
13     std::cin >> n;
14     std::cin.ignore(1024, '\n');
15     for (int i = 0; i < n; i++) {
16         std::pair<std::vector<int>, bool> as;
17         as.second = false;
18         std::getline(std::cin, line);
19         std::stringstream ss(line);
20         while (ss >> a)
21             as.first.push_back(a - 1);
22         al.push_back(as);
23     }
```

```
24
25      return al;
26  }
27
28  bool bpp(std::vector<std::pair<std::vector<int>, bool>>& al,
29      std::vector<int>& path, int v)
30  {
31      al[v].second = true;
32      path.push_back(v);
33      if (v == (int)al.size() - 1)
34          return false;
35      for (auto& a : al[v].first) {
36          if (!al[a].second) {
37              if (!bpp(al, path, a))
38                  return false;
39              path.push_back(v);
40          }
41      }
42      return true;
43  }
44
45  std::vector<int> search(std::vector<std::pair<std::vector<int>, bool>>& al) {
46      std::vector<int> path;
47      bpp(al, path, 0);
48      return path;
49  }
50
51  int main() {
52      int T;
53
54      std::cin >> T;
55
56      /* leer grafo */
57      for (int i = 0; i < T; i++) {
58          auto al = read_graph();
59
60          auto path = search(al);
61
62          std::cout << "Caso " << i + 1 << std::endl;
63
64          if (!al.back().second)
65              std::cout << "INFINITO" << std::endl;
66          else {
67              std::cout << path.size() << std::endl;
68              for (int p : path)
69                  std::cout << p + 1 << std::endl;
70          }
71      }
72  }
73
```

2.3. 405

2.3.1. Analisis, diseño y eficiencia

El grafo se representa como un vector de pares de nodos, y una correspondiente lista de tiempos de reunión. En un vector aparte de guarda el estado de nada nodo (si se ha enterado de la noticia).

En la función principal se leen los datos de stdin, se comprobaría si tiene solución (si todos los nodos están conectados); a continuación, mi solución consiste en simular cada instante de tiempo, el estado de cada nodo.

Al inicio, solo el primer nodo es actualizado. Por cada instante se iteran todos los pares de nodos. Si este coincide con una reunión, y uno de los nodos está actualizado, se actualiza el otro nodo. Y cada instante se comprueba si todos los nodos han sido actualizados. En ese caso la función acaba y retorna cuantos instantes se tardó en llegar.

Este método ofrece ventajas como computar en que instante cada nodo es actualizado, pero a la vez no es muy eficiente en responder al problema dado.

Honestamente no se me ocurrió un algoritmo de grafos apropiado, simplemente simulando la evolución fué la unica forma que se me ocurrió de resolver el ejercicio, y aun así costó hacer que pasase todos los tests, por alguna razón al final fallaba solo un pequeño numero de ellos.

En este caso, la complejidad temporal es directamente la solución, y proporcional a n*a pues se iteran n nodos para comprobar si se han actualizado, y se simulan a aristas (pares) y quedaría tal que O(x*n*a) siendo x la solución, en cuantos instantes se actualizan todos los nodos.

2.3.2. Listado del código

405.cpp 129L 3344B

```
 1 #include <iostream>
 2 #include <vector>
 3 #include <set>
 4 #include <algorithm>
 5
 6 // root node = 1
 7 // day = 100 slots
 8
 9 constexpr int slotsperday = 100;
10
11 struct pair_t {
12     int nod1, nod2;
13     std::vector<int> meetslots;
14 };
15
16 std::vector<pair_t> read_graph(int& nnodes) {
17     int npairs, nmeets, meetslot;
18     std::vector<pair_t> pairs;
19     std::cin >> nnodes >> npairs;
20     for (int i = 0; i < npairs; i++) {
21         pair_t pair = { };
22
23         std::cin >> pair.nod1 >> pair.nod2 >> nmeets;
```

```
24
25          for (int j = 0; j < nmeets; j++) {
26              std::cin >> meetslot;
27              pair.meetslots.push_back(meetslot);
28          }
29
30          pairs.push_back(pair);
31      }
32
33      return pairs;
34  }
35
36  bool checkisolated(const std::vector<pair_t>& pairs, int nnodes) {
37      std::set<int> connnodes;
38      connnodes.insert(1); // root node
39      int p0 = 0, p1 = 1;
40      while (p0 != p1) {
41          for (int nod = 2; nod <= nnodes; nod++) { // rests of nodes
42              if (connnodes.find(nod) != connnodes.end())
43                  continue;
44              for (const auto& pair : pairs) {
45                  if (connnodes.find(pair.nod1) != connnodes.end() ||
46                      connnodes.find(pair.nod2) != connnodes.end())
47                  {
48                      connnodes.insert(pair.nod1);
49                      connnodes.insert(pair.nod2);
50                  }
51              }
52          }
53
54          p0 = p1;
55          p1 = connnodes.size();
56      }
57
58      return (int)connnodes.size() != nnodes;
59  }
60
61  bool checkupdate(const std::vector<bool>& nodeupdated) {
62      return std::find(nodeupdated.begin(), nodeupdated.end(), false)
63          == nodeupdated.end();
64  }
65
66  int sim(std::vector<pair_t>& pairs, int nnodes) {
67      int slot = 0;
68      std::vector<bool> nodeupdated(nnodes, false);
69      nodeupdated[0] = true; // root
70
71      while (!checkupdate(nodeupdated)) {
72          if (slot > 10000)           // todo mal, esto mal, lo sé, fatal
73              return -1;
74          int c = -1;
75          while (c != 0) {
76              c = 0;
77              for (const auto& pair : pairs) {
78                  if ((nodeupdated[pair.nod1-1] || nodeupdated[pair.nod2-1]
79                      && std::find(pair.meetslots.begin(),
80                          pair.meetslots.end(), slot % slotsperday)
81                      != pair.meetslots.end())
82                  {
```

```
 83                     if (nodeupdated[pair.nod1-1] !=
 84                         nodeupdated[pair.nod2-1])
 85                     {
 86                         c++;
 87                         nodeupdated[pair.nod1-1] =
 88                             nodeupdated[pair.nod2-1] = true;
 89                     }
 90                 }
 91             }
 92         }
 93
 94 #ifdef _DEBUG_
 95         std::cout << slot << "\t";
 96         for (const auto& n : nodeupdated)
 97             std::cout << n << " ";
 98         std::cout << std::endl;
 99 #endif
100
101         slot++;
102     }
103
104     return slot - 1;
105 }
106
107 int main() {
108     int cases;
109
110     std::cin >> cases;
111
112     for (int i = 0; i < cases; i++) {
113         int nnodes;
114         auto pairs = read_graph(nnodes);
115         /*if (checkisolated(pairs, nnodes)) { // no funciona, no hay tiempo
116             std::cout << "-1" << std::endl;
117             continue;
118         }*/
119
120 #ifdef _DEBUG_
121         for (const auto& p : pairs)
122             std::cout << "<" << p.nod1 << ", " << p.nod2 << ">, ";
123         std::cout << std::endl;
124 #endif
125
126         std::cout << sim(pairs, nnodes) << std::endl;
127     }
128 }
129
```

3. Conclusión

Los algoritmos sobre grafos son soberanamente antiintuitivos. Ayudaría tener alguna forma de visualizar todos estos algoritmos, ya que por lo menos en mi caso, entiendo mejor las cosas visualmente. Entenderlos es muy complicado, pero mediante estos ejercicios se puede llegar a desarrollar comprensión y practica para saber como funcionan.

Algunos de los algoritmos aplicados podrian ser ineficientes para el caso, y otros algoritmos serían mas optimos.

Al final, esta practica a tomado un tiempo y esfuerzo no trivial, sobre que se empezara tarde (24 dic por la tarde), ha tomado la mayoria de los dias hasta hoy, 27 de diciembre. Si suponemos 8 horas efectivas, 32 horas total. Me he atascado bastante, ya que el tema de los grafos no es algo que se interioice facilmente, y no lo hemos trabajado lo suficiente ni en esta ni en otras asignaturas.

Intenté hacer otros ejercicios como el 407, el 411, etc, pero no logré desatascarme de ellos. Al final solo completé el 402, 403 y 405. De haber tenido mas tiempo habría intentado el 491.