IBERO
CIUDAD DE MÉXICO ®

TSU en Software

Tarea 2: Implementación del algoritmo Dijkstra en pseudocódigo y en C+++

Desarrollo de Aplicaciones II

Carlos Araiza Dionicio

12-02-2021

Implementación del algoritmo Dijkstra

El algoritmo de Dijkstra puede implementarse de diferentes formas en función al tipo de estructuras de datos elegida, sin embargo, en su forma estándar sigue la siguiente estructura:

Pseudocódigo:

```
funcion dijkstra(Grafo, Origen)
    para cada vertice V en G
        fistancia[V] <- infinito
        anterior[V] <- NULL
        If V != Origen,
            agregar V a la priority queue Q
    distance[Origen] <- 0

    Mientras Q no está vacío
        U <- Sacar el valor MIN de Q
        por cada vecino no visitado V de U
            distanciaTemporal <- distancia[U] + peso_camino(U, V)
            Si distanciaTemporal < distancia[V]
                distancia[V] <- distanciaTemporal
                anterior[V] <- U
    regresar distancia[], previo[]
```

En código en C++ sería:

Código fuente:
https://github.com/iCharlieAraiza/DDAII/blob/main/Tarea%202/dijkstra.cpp

```
#include <iostream>
#include <vector>

#define INT_MAX 10000

using namespace std;

void DijkstrasTest();
```

```
int main() {
  DijkstrasTest();
  return 0;
}

class Nodo;
class Arista;

void Dijkstras();
vector<Nodo*> * NodosAdyacentesRestantes(Nodo *nodo);
Nodo * ExtractSmallest(vector<Nodo*>& nodos);
int Distancia(Nodo *nodo1, Nodo *nodo2);
bool Contiene(vector<Nodo*>& nodos, Nodo *nodo);
void ImprimirMinimoCaminoA(Nodo *destino);

vector<Nodo*> nodos;
vector<Arista*> aristas;

class Nodo {
   public:
  Nodo(char id)
    : id(id), prev(NULL), distanciaDesdeInicio(INT_MAX) {
    nodos.push_back(this);
  }

   public:
  char id;
  Nodo* prev;
  int distanciaDesdeInicio;
};

class Arista {
   public:
  Arista(Nodo *nodo1, Nodo *nodo2, int distancia)
    : nodo1(nodo1), nodo2(nodo2), distancia(distancia) {
    aristas.push_back(this);
  }
```

```
  bool Conecta(Nodo *nodo1, Nodo *nodo2) {
    return (
      (nodo1 == this->nodo1 &&
       nodo2 == this->nodo2) ||
      (nodo1 == this->nodo2 &&
       nodo2 == this->nodo1));
  }

  public:
  Nodo* nodo1;
  Nodo* nodo2;
  int distancia;
};

////////////////////
void DijkstrasTest() {
  Nodo* a = new Nodo('a');
  Nodo* b = new Nodo('b');
  Nodo* c = new Nodo('c');
  Nodo* d = new Nodo('d');
  Nodo* e = new Nodo('e');
  Nodo* f = new Nodo('f');
  Nodo* g = new Nodo('g');

  Arista* e1 = new Arista(a, c, 1);
  Arista* e2 = new Arista(a, d, 2);
  Arista* e3 = new Arista(b, c, 2);
  Arista* e4 = new Arista(c, d, 1);
  Arista* e5 = new Arista(b, f, 3);
  Arista* e6 = new Arista(c, e, 3);
  Arista* e7 = new Arista(e, f, 2);
  Arista* e8 = new Arista(d, g, 1);
  Arista* e9 = new Arista(g, f, 1);

  a->distanciaDesdeInicio = 0;  // set start nodo
  Dijkstras();
  ImprimirMinimoCaminoA(f);
```

```
}



void Dijkstras() {
  while (nodos.size() > 0) {
    Nodo* minimo = ExtractSmallest(nodos);
    vector<Nodo*>* nodosAdyacentes =
      NodosAdyacentesRestantes(minimo);

    const int size = nodosAdyacentes->size();
    for (int i = 0; i < size; ++i) {
      Nodo* adyacente = nodosAdyacentes->at(i);
      int distancia = Distancia(minimo, adyacente) +
              minimo->distanciaDesdeInicio;

      if (distancia < adyacente->distanciaDesdeInicio) {
        adyacente->distanciaDesdeInicio = distancia;
        adyacente->prev = minimo;
      }
    }
    delete nodosAdyacentes;
  }
}

Nodo * ExtractSmallest(vector<Nodo*>& nodos) {
  int size = nodos.size();
  if (size == 0) return NULL;
  int minPosicion = 0;
  Nodo* minimo = nodos.at(0);
  for (int i = 1; i < size; ++i) {
    Nodo* actual = nodos.at(i);
    if (actual->distanciaDesdeInicio <
      minimo->distanciaDesdeInicio) {
      minimo = actual;
      minPosicion = i;
    }
  }
```

```
  nodos.erase(nodos.begin() + minPosicion);
  return minimo;
}


vector<Nodo*> * NodosAdyacentesRestantes(Nodo *nodo) {
  vector<Nodo*>* nodosAdyacentes = new vector<Nodo*>();
  const int size = aristas.size();
  for (int i = 0; i < size; ++i) {
    Arista* arista = aristas.at(i);
    Nodo* adyacente = NULL;
    if (arista->nodo1 == nodo) {
      adyacente = arista->nodo2;
    } else if (arista->nodo2 == nodo) {
      adyacente = arista->nodo1;
    }
    if (adyacente && Contiente(nodos, adyacente)) {
      nodosAdyacentes->push_back(adyacente);
    }
  }
  return nodosAdyacentes;
}

// Return distancia between two connected nodos
int Distancia(Nodo *nodo1, Nodo *nodo2) {
  const int size = aristas.size();
  for (int i = 0; i < size; ++i) {
    Arista* arista = aristas.at(i);
    if (arista->Conecta(nodo1, nodo2)) {
      return arista->distancia;
    }
  }
  return -1;
}

bool Contiente(vector<Nodo*>& nodos, Nodo *nodo) {
  const int size = nodos.size();
```

```
  for (int i = 0; i < size; ++i) {
    if (nodo == nodos.at(i)) {
      return true;
    }
  }
  return false;
}


void ImprimirMinimoCaminoA(Nodo *destino) {
  Nodo* prev = destino;
  cout << "Distancia from start: "
     << destino->distanciaDesdeInicio << endl;
  while (prev) {
    cout << prev->id << " ";
    prev = prev->prev;
  }
  cout << endl;
}

vector<Arista*> *AristasAdyacentes(vector<Arista*>& Edges, Nodo *nodo);
void RemoverArista(vector<Arista*>& Edges, Arista *arista);

vector<Arista*> *AristasAdyacentes(vector<Arista*>& aristas, Nodo *nodo) {
  vector<Arista*>* aristasAdjacentes = new vector<Arista*>();

  const int size = aristas.size();
  for (int i = 0; i < size; ++i) {
    Arista* arista = aristas.at(i);
    if (arista->nodo1 == nodo) {
      cout << "adyacente: " << arista->nodo2->id << endl;
      aristasAdjacentes->push_back(arista);
    } else if (arista->nodo2 == nodo) {
      cout << "adyacente: " << arista->nodo1->id << endl;
      aristasAdjacentes->push_back(arista);
    }
  }
```

```
  return aristasAdjacentes;
}

void RemoverArista(vector<Arista*>& aristas, Arista *arista) {
  vector<Arista*>::iterator it;
  for (it = aristas.begin(); it < aristas.end(); ++it) {
    if (*it == arista) {
      aristas.erase(it);
      return;
    }
  }
}
```

Referencias

- Dijkstra's Algorithm. (2019). Programiz. https://www.programiz.com/dsa/dijkstra-algorithm
- Wikipedia. (2021, 7 febrero). Algoritmo de Dijkstra. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra