Trabalho 02 com Algoritmo de Dijkstra

Lizandra de Sousa Costa, Lucas Alves da Costa

Julho 2018

1 Introdução

Este trabalho consiste em explorar a classe de problemas de origem-destino e encontrar o menor caminho entre cidades da Europa na prática, onde apresenta a descrição do algoritmo de Dijkstra assim como a determinação do caminho mais curto entre as cidades específicas.

O problema constitui-se no deslocamento de uma cidade ate outra com isso, ela dispõe de várias estradas, passando por diversas cidades ate chegar na cidade esperada.Para que o menor caminho seja encontrado o algoritmo de Dijkstra é aplicado devido oferece uma trajetória com o menor caminho.

A partir da escolha de uma cidade inicial para a busca, o algoritmo calcula o custo mínimo deste cidade para todas as demais cidades do arco.O algoritmo é usado sobre grafos orientados, e admite que todas as arestas(caminhos) possuem pesos não negativo.As distancias entre as vértice(cidades) são os pesos das arestas e através desses pesos os possíveis caminhos pode ser calculados.

2 Grafo na resolução do trabalho

Conceitualmente, "um grafo consiste em um conjunto de nós - ou vértices - e um conjunto de arcos - ou arestas -" (TENENBAUM et. al., 1995), sendo uma das formas de representar estruturas de dados computacionalmente. A teoria dos grafos, como é mais conhecida, tem importância ímpar na representação de problemas como fluxo de rede, roteamento e caminho mínimo (RABUSKE, 1995).

Segundo Tenenbaum et al (1995), em um grafo um vértice é considerado adjacente a um outro vértice se entre estes existir a incidência de uma aresta, sendo um vértice denominado sucessor e o outro predecessor. A relação em um conjunto qual quer consiste em "uma sequencia qualquer de pares ordenados de elementos" (TENENBAUM et al, 1995).

Outras aplicações incluem quaisquer problemas envolvendo redes ou grafos em que se tenha grandezas(distâncias, tempo, perdas, ganhos, despesas) que se acumulem linearmente ao longo do percurso da rede

3 Algoritmo de menor caminho

Os algoritmos de menor caminho - ou caminho mínimo - são aplicados sobre grafos ponderados, em que se deseja achar o menor caminho entre dois nós. O menor caminho consiste na soma dos pesos das arestas de um grafo a partir de um determinado vértice (GUIMARÃES, 2004).

Dado o mapa de tais cidades, contendo as distâncias entre cidades, qual o menor caminho entre quaisquer cidades?

O problema de menor caminho no grafo consiste em determinar o menor caminho entre a cidade inicial e todos as demais cidades. Através de seu peso que posteriormente e somado com o peso de seu vértice adjacente é possível encontrar o seu menor caminho.

Algumas variações do problema no trabalho:
– Menor caminho com destino único.
– Menor caminho entre um par de vértices.
– Menor caminho entre todos os pares de vértices.

4 Algoritmo de Dijkstra

O algoritmo de Dijkstra foi criado pelo cientista da computação holandês Edsger Dijkstra em 1956 e em 1959 foi publicado, soluciona o problema do caminho mais curto num grafo dirigido ou não dirigido com arestas de peso não negativo. Seu tempo computacional é *O([m+n]log n)* onde m é o número de arestas e n é o número de vértices.

O algoritmo de Dijkstra também é conhecido como um "algoritmo guloso" devido sempre escolher o vértice mais leve ou mais próximo ao vértice a que faz adjacência, além de ser um algoritmo que percorre todos os vértices de

um grafo o, a fim de se conhecer à distância do vértice fonte a todos os outros vértices do grafo. O algoritmo busca resolver o problema do menor percurso entre os vértices de um grafo, com origem única, em grafos ponderados no qual os pesos das arestas são não-negativos (PREISS, 2000). O algoritmo percorre todos os vértices do grafo a partir de um determinado vértice inicial ou fonte. "A característica essencial do algoritmo de Dijskstra é a ordem na qual os caminhos são determinados: Os caminhos são encontrados na ordem dos comprimentos ponderados, começando pelo mais curto e prosseguindo até o mais longo" (PREISS, 2000).

5 Aplicação do algoritmo na solução do problema

Para implementar o algoritmo de Djikstra foi utilizada a linguagem de programação C, o seguinte código está contido os pontos principais para implementação do algoritmo. Para implementar o algoritmo de Djikstra é necessário montarmos um grafo, para isso utilizamos o grafo abaixo que está contido dentro da função principal **main()**. Esse grafo é feito usando a estrutura de pilha dinâmica para armazenar os vértices e suas adjacências, neste projeto iremos usar um grafo de seis vértices que é a quantidade equivalente ao numero de cidades propostas no trabalho, cada cidade será representada por um vértice e as arestas que tornam as vértices adjacentes entre si são representações da distancia em km que separa uma cidade a outra. Tendo em vista as adjacências dos vértices sabemos que cidade possui um ou mais caminhos em relação as outras. Para montarmos o grafo utilizamos a função **add aresta()** que é responsável por iniciar a ligação entre os vetrices.

```
1  int main(int argc, char *argv[]) {
2
3    printf("\t\t\t\t%cCIDADES DA EUROPA%c\n",186,186);
4    printf("\t\t\t\t%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c
       %c\n\n\n",
5    254,254,254,254,254,254,254,254,254,254,254,
6    254,254,254,254,254,254,254,254);
7
8        grafo_t * g = calloc(1, sizeof(grafo_t));
9
```

```
        add_aresta(g, 0, 1, 1277);
        add_aresta(g, 0, 2, 1659);

        add_aresta(g, 1, 0, 1277);
        add_aresta(g, 1, 2, 656);
        add_aresta(g, 1, 3, 510);

        add_aresta(g, 2, 0, 1659);
        add_aresta(g, 2, 1, 656);
        add_aresta(g, 2, 3, 817);
        add_aresta(g, 2, 5, 683);

        add_aresta(g, 3, 1, 510);
        add_aresta(g, 3, 2, 817);
        add_aresta(g, 3, 4, 655);
        add_aresta(g, 3, 5, 876);


        add_aresta(g, 4, 3, 655);
        add_aresta(g, 4, 5, 350);

        add_aresta(g, 5, 2, 683);
        add_aresta(g, 5, 3, 876);
        add_aresta(g, 5, 4, 350);


        menuDistancia(g);

        return 0;
}
```

Esta função tem como primeiro argumento uma estrutura do tipo grafo, o segundo e o terceiro referem-se ao nome ou identificação de cada vértice, o ultimo argumento é a distancia entre o vértice do segundo argumento e o terceiro. vamos analisar agora como essa função cria um vértice.

```
void add_aresta(grafo_t * g, int a, int b, int w) {
        add_vertex(g, a);
        add_vertex(g, b);

```

```
5           vertex_t *v = g->vertices[a];
6
7           if (v->aresta_len >= v->aresta_size) {
8                   v->aresta_size = v->aresta_size ? v->
                       aresta_size * 2 : 4;
9                   v->arestas = realloc(v->arestas, v->
                       aresta_size * sizeof (aresta_t *));
10          }
11
12          aresta_t * e = calloc(1, sizeof(aresta_t));
13          e->vertex = b;
14          e->peso = w;
15
16          v->arestas[v->aresta_len++] = e;
17  }
18
19  heap_t * create_heap(int n) {
20          heap_t * h = calloc(1, sizeof(heap_t));
21          h->data = calloc(n+1, sizeof(int));
22          h->prio = calloc(n+1, sizeof(int));
23          h->index = calloc(n, sizeof(int));
24
25          return h;
26  }
```

Note que essa função utiliza como sequencia a função **add vertex** que usa os três primeiros argumentos da função anterior para criar os vértices com os nomes passados como parâmetro, neste caso são usados números para identificar os vértices, em seguida é atribuído um peso entre essas vértices criadas. Para armazenar as rotas entre os vértices e as distâncias, utilizamos a função **create heap** que cria uma pilha dinâmica. Veja abaixo a função responsável por criar os vértices.

```
1   void add_vertex(grafo_t * g, int i) {
2           if (g->vertices_size < i + 1) {
3                   int size = g->vertices_size * 2 > i ? g
                        ->vertices_size * 2 : i + 4;
4                   g->vertices = realloc(g->vertices, size*
                        sizeof(vertex_t *));
5
```

```
6                   int j;
7                   for (j = g->vertices_size; j < size; j
                        ++) {
8                         g->vertices[j] = NULL;
9                   }
10
11                  g->vertices_size = size;
12          }
13
14          if (!g->vertices[i]) {
15                  g->vertices[i] = calloc(1, sizeof(
                        vertex_t));
16                  g->vertices_len++;
17          }
18  }
```

Já temos nossos vértices criados e ligados por arestas, agora vamos percorrê-los afim de encontrar a menor rota entre uma origem x e um destino y, para isso utilizaremos o tão esperado algoritmo de Djikstra. Veja abaixo sua implementação e as funções que acompanham sua funcionalidade.

```
1   void dijkstra(grafo_t * g, int a, int b) {
2       printf("Calculando rota com algoritmo de Djikstra:\n
            ");
3           int i, j;
4           for (i = 0; i < g->vertices_len; i++) {
5                   vertex_t * v = g->vertices[i];
6                   v->dist = INT_MAX;
7                   v->anterior = 0;
8                   v->visited = 0;
9           }
10          vertex_t * v = g->vertices[a];
11          v->dist = 0;
12          heap_t * h = create_heap(g->vertices_len);
13          push_heap(h, a, v->dist);
14          while (h->len) {
15                  i = pop_heap(h);
16
17                  if (i == b)
18                          break;
```

```
19
20                  v = g->vertices[i];
21                  v->visited = 1;
22                  for (j = 0; j < v->aresta_len; j++) {
23                          aresta_t * e = v->arestas[j];
24                          vertex_t * u = g->vertices[e->
                               vertex];
25                          if (!u->visited && v->dist + e->
                               peso <= u->dist) {
26                                  u->anterior = i;
27                                  u->dist = v->dist + e->
                                       peso;
28                                  push_heap(h, e->vertex,
                                       u->dist);
29                          }
30                  }
31          }
32  }
```

Essa função recebe como argumento o grafo onde estão nossos vértices e uma variável que será um ponto de partida e outra variável que será o destino da rota a ser traçada. De inicio todos os vértices são taxados com um valor bem alto que representa um valor infinito, a partir do vértice de partida são calculadas as menores rotas para seus vértices adjacentes, apôs escolher o vértice de menor peso o vértice anterior e taxado como visitado, cada vértice visitado é armazenado na pilha pela função **push heap()**, após chegar ao vértice de destino teremos um custo x, em seguida serão verificados todos os vértices que ainda não foram taxados como visitados, após verificar todos esses vértices teremos um custo y, caso o custo até y seja menor que o custo até x então a rota de x será retirada da pilha pela função **pop head()** e será empilhada a rota de y em seu lugar. Esse é o funcionamento deste algoritmo aplicado no problema proposto. Agora que achamos o menor caminho entre os vértices de partida e destino já podemos imprimir a rota correspondente ao menor caminho. Veja abaixo a função responsável por imprimir a rota de menor custo entre os vértices.

```
1  void imprimir_caminho(grafo_t * g, int i) {
2          int n, j;
3          vertex_t *v, *u;
```

```
4           v = g->vertices[i];
5           if (v->dist == INT_MAX) {
6                printf("Sem rota para essa cidade!\n\n");
7                return listaCity();
8           }
9
10          for (n = 1, u = v; u->dist; u = g->vertices[u->
               anterior], n++);
11
12          char * path = malloc(n);
13
14          path[n-1] = 'a' + i;
15
16          for (j = 0, u = v; u->dist; u = g->vertices[u->
               anterior], j++)
17               path[n - j - 2] = 'a' + u->anterior;
18
19
20          for(j=0; j<n; j++){
21              letra_city(&path[j]);
22              if(j<n-1)
23              printf("%c ",175);
24          }
25          printf("\n\nMENOR DISTANCIA DE: "); letra_city(&
               path[0]);
26          printf(" PARA: "); letra_city(&path[n-1]);
27          printf("= %d_km\n\n\n",v->dist);
28
29  }
```

Essa função consiste em imprimir todos os vértices salvos pela pilha, cada vértice tem um numero que a identifica, esse numero será convertido em um nome de cidade para que faça sentido nossa rota imprimida, note que o vetor que recebe as rotas *path[]* recebe os valores de cada vértice e os converte em letras do alfabeto, tento em vista seis vértices ao invés de termos *0,1,2,3,4,5* temos *a,b,c,d,e,f*. Cada vértice representa uma cidade da Europa, então vamos converter as letras do vetor de rotas em nomes de cidades com a seguinte função:

```
1   void letra_city(const char *G){
```

```
2       switch(G[0]){
3             case 'a':
4             printf("(Madrid-Espanha) ");
5             break;
6
7             case 'b':
8              printf("(Paris-Fran%ca) ",135);
9              break;
10
11              case 'c':
12               printf("(Zurique-Sui%ca) ",135);
13               break;
14
15               case 'd':
16                 printf("(Amsterd%c-Holanda) ",198);
17                 break;
18
19                 case 'e':
20                   printf("(Berlin-Alemanha) ");
21                   break;
22
23                    case 'f':
24                     printf("(Praga-Republica_Tchequia) "
                          );
25                     break;
26
27                      default:
28                         printf("City not identified!!");
29                         break;
30      }
31  }
32  void listaCity(){
33   printf("\n_____________________________________\n");
34   printf("%cId %c Cidade                          %c\n"
        ,186,186,186);
35   printf("-------------------------------------\n");
36   printf("|0 -| Madrid - Espanha.              |\n");
37   printf("-------------------------------------\n");
38   printf("|1 -| Paris - Fran%ca.               |\n",135);
39   printf("-------------------------------------\n");
```

```
40  printf("|2 -| Zurique - Sui%ca.              |\n",135);
41  printf("-----------------------------------\n");
42  printf("|3 -| Amsterd%c - Holanda.          |\n",198);
43  printf("-----------------------------------\n");
44  printf("|4 -| Berlin - Alemanha.           |\n");
45  printf("-----------------------------------\n");
46  printf("|5 -| Praga - Republica Tchequia. |\n");
47  printf("-----------------------------------\n");
48 }
```

Note que cada letra corresponde a um numero de zero a cinco, ou seja o vértice zero corresponde a letra 'a' e o vértice um corresponde a letra 'b' e assim por diante. Em sequencia da função de conversão de letras em nomes temos uma tabela com a lista de vértices que podemos traçar rotas entre si, como por exemplo; Do vértice *0* para o vértice *1* seria o mesmo de dizer; De Madrid-Espanha para Paris-França. Uma pequena observação, os parâmetros das funções *printf()* são apenas para utilizar caracteres especiais da tabela ASCII.

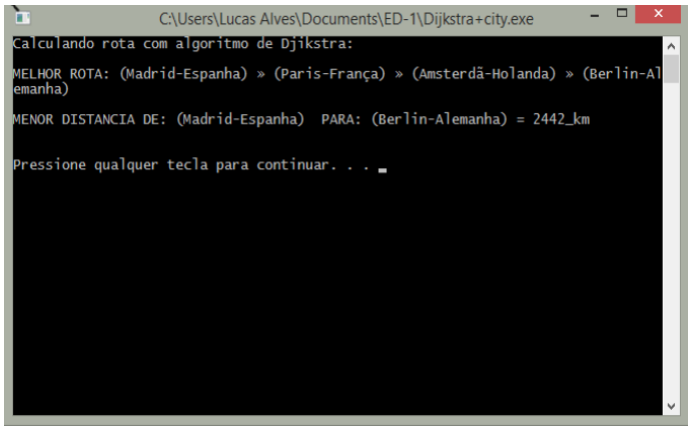

Figura 1: Resultado final

6 Conclusão

O objetivo do trabalho proposto em definir menor rota entre cidades europeias, em que dado um conjunto de localidades a serem visitadas, foi bem

solucionado utilizando o algoritmo de Dijkstra pois mostrou-se eficiente ferramenta para a valoração das distâncias entre pontos referenciados, bem como suas rotas. A sua aplicação no método possibilitou a análise de rotas entre um ponto de origem e destinos pré-definidos, relevantes ao problema descrito. O método de busca exaustiva possibilita a identificação da rota mínima, dados as distâncias através do algoritmo. Ao fim, o método de Dijkstra é utilizado novamente para identificar a rota total, passando por todos os pontos referenciados. Assim, o trabalho proposto parte da análise de pontos adjacentes até a identificação de uma estimativa de rota completa de distância mínima para percorrer os pontos considerados.

7 Referências

Algoritmo de Dijkstra - Universidade de São Paulo - IME/USP
Departamento de informática e estatística.
Viva o Linux
Wikipedia - Algoritmo de Dijkstra
Programação descomplicada - Grafos
GitHub
Universidade federal do Paraná Teoria dos Grafos - Guimarães, José de O.
Estruturas de Dados usando C, tradução - TENENBAUM, Aaron M., AUGENSTEIN, Moshe J.

Livros.
Titulo: Estruturas de Dados e Algoritmos – Padrões de Projetos Orientados a Objetos com Java,
Autor: PREISS, Bruno R
Ano: 2000,
publisher: Campus.

Rabuske1995inteligencia,
Titulo: Inteligência Artificial,
Autor: RABUSKE, Renato A.
Ano: 1995,
publisher: Editora da UFSC