Trabalho Prático de
Projeto e Análise de Algoritmos

Gabriel Souza e Luiza Ávila

Introdução

Por meio desse documento, visamos explicar a realização do trabalho prático proposto no início do semestre. O trabalho continha três problemas para serem implementados: o algoritmo guloso para resolver o problema do caminho mais curto em um grafo com peso não negativo, algoritmo para determinar se existe algum ciclo negativo no grafo e o algoritmo de programação dinâmica para resolver o problema do caminho mais curto em um grafo com aresta negativa, mas sem ciclo negativo.

O problema do caminho mais curto consiste na construção de um grafo e do desejo de encontrar o caminho em que a soma das arestas de um vértice *a* para um vértice *b* seja a menor possível. Deveremos usar o algoritmo guloso de Dijkstra para resolver o problema. A ideia básica do Dijkstra é percorrer o grafo armazenando em um conjunto os candidatos que foram checados e aprovados. O algoritmo considera um conjunto S de menores caminhos, iniciado com um vértice inicial I. A cada passo do algoritmo busca-se nas adjacências dos vértices pertencentes a S aquele vértice com menor distância relativa a I e adiciona-o a S e, então, repetindo os passos até que todos os vértices alcançáveis por I estejam em S. Arestas que ligam vértices já pertencentes a S são desconsideradas.

O problema dos ciclos negativos e o caminho mais curto sem eles pode ser resolvido utilizando o mesmo algoritmo, o Bellman-Ford. Assim como o algoritmo de Dijkstra, o algoritmo de Bellman-Ford utiliza a técnica de relaxamento, ou seja, realiza sucessivas aproximações das distâncias até finalmente chegar na solução. A principal diferença entre Dijkstra e Bellman-Ford é que no algoritmo de Dijkstra é utilizada uma fila de prioridades para selecionar os vértices a serem relaxados, enquanto o algoritmo de Bellman-Ford simplesmente relaxa todas as arestas.

Objetivamos no trabalho encontrar a solução ótima para os três problemas. Para cada um dos problemas, explicaremos de forma sucinta o funcionamento algoritmo, analisaremos a ordem de complexidade e mostraremos os casos testes e as conclusões que retiramos.

Implementação

Primeiro problema:

Algoritmo guloso para resolver o problema do caminho mais curto em um grafo com peso não negativo (Algoritmo de Dijkstra).

Para resolver esse problema usamos o algoritmo guloso do Dijkstra. Para cada vértice $u \in V$, manteremos um atributo *d[v]* que será um limitante superior para um caminho mínimo de *s a v*. O algoritmo mantém um conjunto *S* que contém os vértices com os caminhos mínimos calculados até o momento. A cada iteração o algoritmo seleciona um novo vértice *v* para ser incluído em *S*, tal que *v* e escolhido entre os vértices de $V - S$ com menor valor de *d[v]*. O vértice *v* e incluído em *S* e todas as arestas que saem de *v* são processadas pela rotina *Relax* (relaxamento: o processo de relaxar uma aresta *(u,v)* consiste em testar se podemos melhorar o caminho mais curto para *v* encontrado até agora pela passagem através de *u* e, neste caso, atualizar *d[v]* e *π[v]*).

<u>Pseudoalgoritmo:</u>

procedimento DIJKSTRA(*G, w, s*) >**Entrada**: *G = (V, E)*, conjunto de pesos das arestas (*w*) e o vértice inicial *s*

$S \leftarrow \infty$

$Q \leftarrow V$

enquanto *(* ***faça*** $Q \neq \emptyset$ *)*

u ← ExtraiMin(Q) *>Em relação a d[]*

$S \leftarrow S \cup \{u\}$

para cada *vértice* $v \in$ *Adj(u) faça*

> Adj(u) é a lista de arestas para o vértice u

Relax(u,v,w)

fim para cada

fim enquanto

fim procedimento

Segundo problema & Terceiro problema: Algoritmo para determinar se existe um ciclo negativo em um grafo. Algoritmo de programação dinâmica para resolver o problema do caminho mais curto em um grafo com aresta negativa, mas sem ciclo negativo.

Para esses problemas usamos o algoritmo de Bellman-Ford. O algoritmo recebe um grafo orientado *(G, w)* (possivelmente com arestas de peso negativo) e um vértice origem *s* de *G*. Ele devolve um valor booleano *FALSE* se existe um ciclo negativo atingível a partir de *s*, ou *TRUE* e neste caso devolve também uma Árvore de Caminhos Mínimos com raiz *s*. A ideia do algoritmo Bellman-Ford é avaliar repetidamente as equações utilizando os valores da iteração precedente.

Passo 0 (Inicialização): Dado o vértice de origem s, inicialize as distâncias como:

$$d^0[k] = \begin{cases} 0 \text{ se } k = s. \\ \infty \text{ caso contrário.} \end{cases}$$

e o contador de iterações como $t = 1$.

Passo 1 (Atualização): Para cada vértice $k \neq s$ faça:

$$d^t[k] = \min_i \{ d^{t-1}[i] + c_{i,k} : (i, k) \text{ existe} \}$$

Se $d^t[k] < d^{t-1}[k]$ faça também $p[k] = \arg\min_i \{ d^{t-1}[i] + c_{i,k} : (i, k) \text{ existe} \}$

Passo 3 (Critérios de Parada): Termine se $d^t[k] = d^{t-1}[k]$ para todos os vértices k ou se $t = n$ (iterações = no. de vértices do grafo). Caso contrário faça $t = t + 1$ e volte para o Passo 1.

- Os valores *d[k]* representam os custos ou distâncias dos caminhos mais curtos a partir da origem até qualquer vértice *k*.
- Os caminhos em si podem ser recuperados através dos valores auxiliares *p[k]* também computados no passo 1 do algoritmo.
- Para um dado vértice *k*, *p[k]* é simplesmente o índice ou rótulo do vértice que precede *k* no caminho ótimo da origem *s* até *k*.
- Para obter o caminho ótimo de *s* até *k* basta portanto seguir os índices *p* a partir de *k* retroativamente até *s*

Se o grafo não contiver diciclos negativos todos os valores terão convergido no máximo após t = n – 1. De fato, se houver um diciclo negativo alcançável a partir do vértice de origem s, então a cada iteração o algoritmo indefinidamente encontrará um caminho de ainda menor custo através do diciclo e os valores nunca irão convergir. Portanto, se algum valor ainda se alterar na iteração t = n, então o grafo necessariamente possui um diciclo negativo. Logo, o algoritmo Bellman-Ford pode também ser utilizado para detectar a presença de diciclos negativos em grafos.

Análise de Complexidade

Ambos os algoritmos têm como operação relevante a comparação entre elementos do array.

Algoritmo de Djikstra:

A fila de prioridade *Q* e mantida com as seguintes operações:

- INSERT
- EXTRACT-MIN e
- DECREASE-KEY (implícito em RELAX).

São executados (no máximo) **|V|** operações EXTRACT-MIN. Cada vértice $u \in V[G]$ é inserido em *S* (no máximo) uma vez e cada aresta *(u, v)* com $v \in Adj[u]$ é examinada (no máximo) uma vez durante todo o algoritmo. Assim, são executados no máximo **|E|** operações DECREASE-KEY.

No total temos **|V|** chamadas a EXTRACT-MIN e **|E|** chamadas a DECREASE-KEY.

Implementando *Q* como um vetor (coloque *d[v]* na posição *v* do vetor), INSERT e DECREASE-KEY gastam tempo **Ө(1)** e EXTRACT-MIN gasta tempo **O(V)**, resultando em um total de $\mathbf{O(V^2+ E) = O(V^2)}$.

Implementando a fila de prioridade *Q* como um min-heap, INSERT, EXTRACT-MIN e DECREASE-KEY gastam tempo **O(lgV)**, resultando em um total de **O(V + E)lg V**.

Usando heaps de Fibonacci (EXTRACT-MIN é **O(lg V)** e DECREASE-KEY é **O(1)**) a complexidade reduz para **O(V lgV + E)**.

Algoritmo de Bellman-Ford:

Atribuição de valores iniciais aos *n* vértices: **O(n)**

A cada iteração, cada uma das *m* arestas direcionadas é verificada uma vez (no vértice de entrada) para atualização de valores: **O(m)**

Como são n iterações: **O(nm + n)** $\Rightarrow$ **O(nm)**

Testes

Para cada um dos algoritmos, 4 testes foram realizados: grafo simétrico, assimétrico, com autoloop, com aresta zero. Como o Dijkstra não funciona com arestas negativas, somente os grafos do algoritmo de Bellman-Ford terão arestas com essa característica, além de também possuir um teste com um grafo com ciclos negativos.

Algoritmo de Djikstra

Grafo Simétrico

```
{{-1,  2, -1, -1,  4},
 { 2, -1, 15,  7, -1},
 {-1, 15, -1,  7, -1},
 {-1,  7,  7, -1,  6},
 { 4, -1, -1,  6, -1}};
```

```
Symmetric graph:
Vertex    Distance
0         0
1         2
2         16
3         9
4         4
The shortest way from 0 to 4 is: 0,4
```

Grafo Assimétrico

```
{{-1,  4, -1, -1,  4},
 { 2, -1, 15,  5, -1},
 {-1, 15, -1,  7, -1},
 {-1,  7,  7, -1,  6},
 { 4, -1, -1,  10, -1}};
```

```
Asymmetric graph:
Vertex   Distance
0        0
1        4
2        16
3        9
4        4
The shortest way from 0 to 4 is: 0,4
```

Grafo com Autoloop

```
{{-1,  4, -1, -1,  4},
 { 2, -1, 15,  5, -1},
 {-1, 15, -1,  7, -1},
 {-1,  7,  7, 8,  6},
 { 4, -1, -1,  10, -1}};
```

```
Autoloop graph:
Vertex    Distance
0         0
1         4
2         16
3         9
4         4
The shortest way from 0 to 4 is: 0,4
```

Grafo com Aresta Zero

```
{{-1,  4, -1, -1,  0},
 { 2, -1, 15,  5, -1},
 {-1, 15, -1,  7, -1},
 {-1,  7,  7, 8,  6},
 { 0, -1, -1,  10, -1}};
```

```
Edge zero graph:
Vertex    Distance
0         0
1         4
2         16
3         9
4         0
The shortest way from 0 to 4 is: 0,4
```

Algoritmo de Bellman-Ford

Grafo Simétrico

```
{{N, 4, N, N, 3},
 {4, N, 9, 5, N},
 {N, 9, N, 7, N},
 {N, 5, 7, N, 1},
 {3, N, N, 1, N}};
```

```
Symmetric graph:
Vertex   Distance
0        0
1        4
2        11
3        4
4        3
The shortest way from 0 to 4 is: 0,4
```

Grafo Assimétrico

```
{{N, -2,  N,  N,  3},
 {4,  N,  9,  5,  N},
 {8,  9,  N,  7,  N},
 {N,  5,  5,  N,  1},
 {3,  N, -1,  6,  N}};
```

```
Asymmetric graph:
Vertex   Distance
0        0
1        -2
2        2
3        3
4        3
The shortest way from 0 to 4 is: 0,4
```

Grafo com Autoloop

```
{{N,  4,  N,  N,  4},
 {2,  N, 15,  5,  N},
 {N, 15,  N,  7,  N},
 {N,  7,  7, -1,  6},
 {4,  N,  N,  10, N}};
```

```
Autoloop graph:
Graph contains negative-weight cycle
Graph contains negative-weight cycle
```

Grafo com Aresta Zero

```
{{N,  4,  N,  N,  0},
 {2,  N, 15,  5,  N},
 {N, 15,  N,  7,  N},
 {N,  7,  7,  8,  6},
 {0,  N,  N, 10,  N}};
```

```
Edge zero graph:
Vertex    Distance
0         0
1         4
2         16
3         9
4         0
The shortest way from 0 to 4 is: 0,4
```

Grafo com Ciclo de Peso Negativo

```
{{N, -4,  2, N, 4},
 {2,  N, -1, 7, N},
 {2, -1,  N, N, N},
 {N,  7,  N, N, 6},
 {4,  N,  N, 6, N}};
```

```
Negative-weight cycle graph:
Graph contains negative-weight cycle
Graph contains negative-weight cycle
```

Conclusão

A partir da nossa análise dos dois algoritmos, conseguimos concluir que o algoritmo de Bellman-Ford é mais genérico que o algoritmo de Djikstra, porém o último é mais rápido. Isso se deve ao fato de que o primeiro algoritmo checa para ver se existem ciclos negativos, enquanto o outro já se assume que não, pois nem arestas negativas existem. Por isso, caso em dúvida sobre o grafo, o melhor a ser usado é o de Bellman-Ford.

A principal dificuldade em relação a implementação foi encontrar um algoritmo que conseguiria realizar o que o Djikstra não consegue, mexer com arestas negativas. Depois de encontrarmos o Bellman-Ford, a seleção dos testes que deveriam ser realizados foi o maior desafio.

Bibliografia

SOUZA, C.C. de, SILVA C.N. da, LEE,O., REZENDE, P.J. de.; **MO417 — Complexidade de Algoritmos I;** Disponível em: <http://www.ic.unicamp.br/~rezende/ensino/mo417/2012s2/Slides/Slides20121017-Cam.Min-4x1.pdf > Acesso em 2 jun. 2019

DELGADO, K.V.; **Problema do caminho mais curto de uma única origem em grafos;** Disponível em: <http://www.each.usp.br/digiampietri/SIN5013/Karina_Aula13AA.pdf > Acesso em 2 jun. 2019

CAMPELLO, R.J.G.B.; **Grafos VI: Grafos Ponderados & Caminhos Mínimos (Bellman-Ford) ;** Disponível em: <http://wiki.icmc.usp.br/images/e/e9/Grafos_VI.pdf > Acesso em: 2 jun. 2019