Geekly articles weekly

Algoritmo de Bellman-Ford

Antecipando o início do curso "Algoritmos para desenvolvedores", eles prepararam outra tradução de um artigo interessante.

Problema : dado um gráfico e o vértice inicial src no gráfico, é necessário encontrar os caminhos mais curtos de src para todos os vértices deste gráfico. O gráfico pode conter arestas com pesos negativos.

Já discutimos o algoritmo de Dijkstra como uma maneira de resolver esse problema. O algoritmo de Dijkstra é um algoritmo ganancioso e sua complexidade é O (VLogV) (usando a pilha de Fibonacci). No entanto, Dijkstra não funciona para gráficos com pesos de arestas negativos, enquanto Bellman-Ford é completamente. O algoritmo Bellman-Ford é ainda mais simples que o algoritmo Dijkstra e é adequado para sistemas distribuídos. Ao mesmo tempo, sua complexidade é *O (VE)* , que é mais do que o indicador para o algoritmo de Dijkstra.

Recomendação : Antes de continuar visualizando a solução, tente praticar a si mesmo.

Algoritmo

A seguir estão as etapas detalhadas.

Entrada : Gráfico e vértice inicial `src` .

Saída : a menor distância para todos os vértices do src. Se ocorrer um

ciclo de peso negativo, as distâncias mais curtas não serão calculadas, uma mensagem será exibida indicando a presença desse ciclo.

1. Nesta etapa, as distâncias do vértice inicial a todos os outros vértices são inicializadas como infinitas, e a distância para o próprio src é assumida como 0. Uma matriz `dist[]` tamanho `|V|` com todos os valores iguais ao infinito, com exceção do elemento `dist[src]` , em que `src` é o vértice original.
2. O segundo passo calcula as distâncias mais curtas. As etapas a seguir devem ser executadas `|V|` -1 vezes, onde `|V|` - o número de vértices neste gráfico.
    - Execute a seguinte ação para cada borda *uv* :

      Se `dist[v] > dist[u] + uv` , atualize `dist[v]`

      ```
      dist [v] = dist [u] + uv
      ```
3. Nesta etapa, é relatado se um ciclo de peso negativo está presente no gráfico. Para cada borda *uv* , faça o seguinte:
    - Se `dist[v] > dist[u] + uv` , o gráfico conterá um ciclo de peso negativo.

A idéia da etapa 3 é que a etapa 2 garanta a menor distância se o gráfico não contiver um ciclo de peso negativo. Se passarmos por todas as arestas novamente e conseguirmos um caminho mais curto para qualquer um dos vértices, isso será um sinal da presença de um ciclo de peso negativo.

Como isso funciona? Como em outras tarefas de programação dinâmica, o algoritmo calcula os caminhos mais curtos de baixo para cima. Primeiro, calcula as distâncias mais curtas, ou seja, caminhos

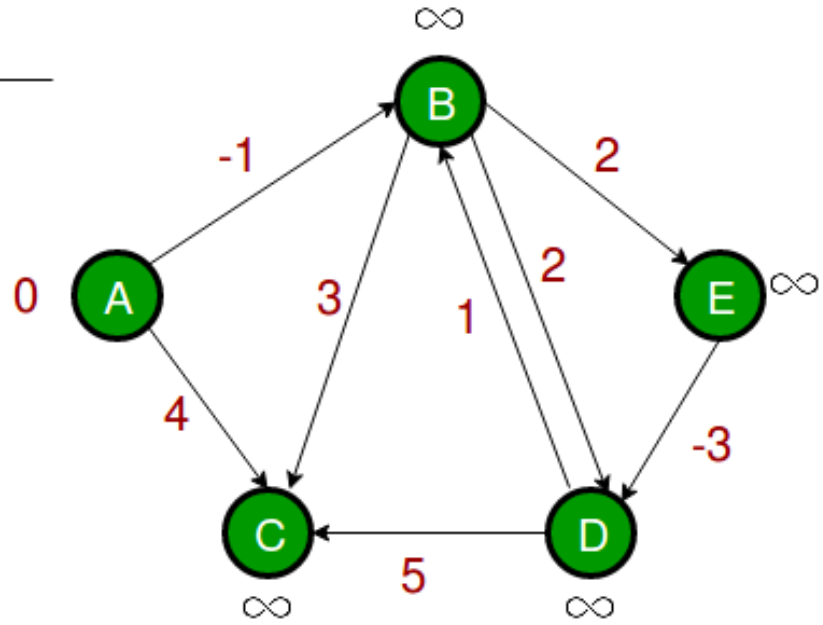
com um comprimento não superior a uma aresta. Em seguida, calcula os caminhos mais curtos com um comprimento não superior a duas arestas e assim por diante. Após a i-ésima iteração do loop externo, são calculados os caminhos mais curtos com um comprimento não superior a *i* arestas. Em qualquer caminho simples, pode haver no máximo *| V | -1* arestas; portanto, o loop externo executa exatamente *| V | -1* vezes. A ideia é que, se calcularmos o caminho mais curto com não mais que *i* arestas, a iteração sobre todas as arestas garantirá o caminho mais curto com não mais que *i + 1* arestas (a prova é bastante simples, você pode consultar esta palestra ou vídeo palestra do MIT)

Exemplo

Vejamos o algoritmo no exemplo de gráfico a seguir. Imagens tiradas daqui .

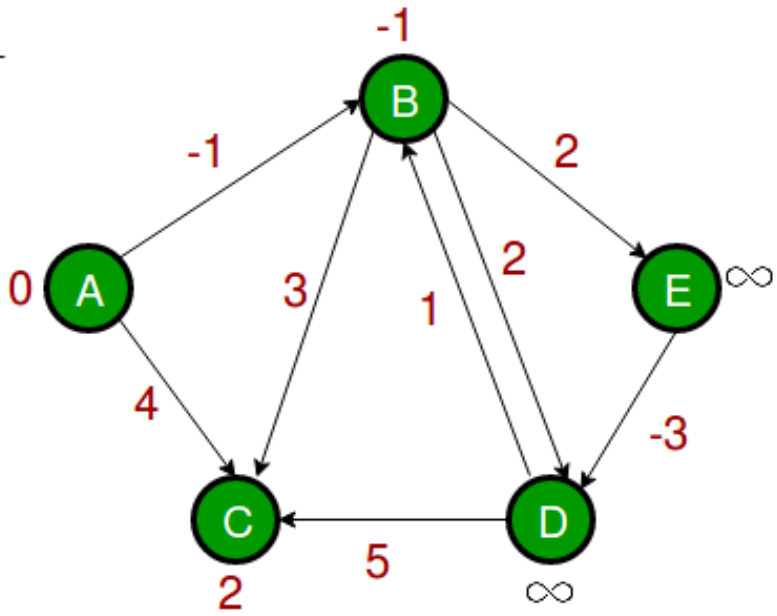
Deixe o vértice inicial ser 0. Tome todas as distâncias como infinitas, exceto a distância para `src` . O número total de vértices no gráfico é 5, portanto, todas as arestas precisam ir 4 vezes.

A	B	C	D	E
0	∞	∞	∞	∞



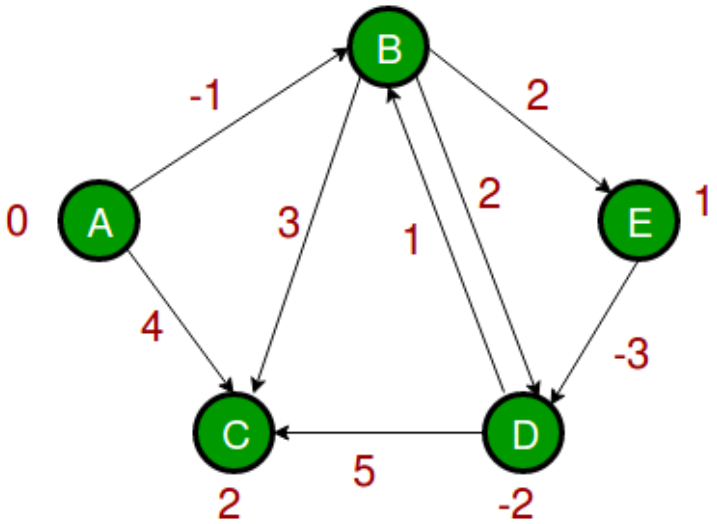
As costelas devem ser trabalhadas na seguinte ordem: (B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D). Temos as seguintes distâncias quando as costelas foram cruzadas. A primeira linha mostra as distâncias iniciais, a segunda linha mostra as distâncias quando as arestas (B, E), (D, B), (B, D) e (A, B) são processadas. A terceira linha mostra a distância de processamento (A, C). A quarta linha mostra o que acontece quando (D, C), (B, C) e (E, D) são processados.

A	B	C	D	E
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞



A primeira iteração garante que todos os caminhos mais curtos não excedam o caminho de 1 borda. Obtemos as seguintes distâncias quando a segunda passagem em todas as arestas é concluída (a última linha mostra os valores finais).

A	B	C	D	E
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞
0	-1	2	∞	1
0	-1	2	1	1
0	-1	2	-2	1



A segunda iteração garante que todos os caminhos mais curtos tenham um comprimento de no máximo 2 arestas. O algoritmo é executado ao longo de todas as bordas mais duas vezes. As distâncias são minimizadas após a segunda iteração, portanto, a terceira e a quarta iterações não atualizam os valores da distância.

Implementação:

```
# Python program for Bellman-Ford's single source # shortest path algor
```

Valores de saída:

```
Vertex   Distance from Source
0                0
1                -1
2                2
3                -2
4                1
```

Notas:

1. Pesos negativos são encontrados em várias aplicações gráficas. Por exemplo, em vez de aumentar o custo de um caminho, podemos nos beneficiar seguindo um caminho específico.
2. O algoritmo Bellman-Ford funciona melhor para sistemas distribuídos (melhor que o algoritmo de Dijkstra). Ao contrário de

Dijkstra, onde precisamos encontrar o valor mínimo de todos os vértices, em Bellman Ford, as arestas são consideradas uma de cada vez.

Exercícios:

1. O algoritmo padrão de Bellman-Ford relata caminhos mais curtos apenas se não tiver ciclos de peso negativo. Modifique-o para que ele relate os caminhos mais curtos, mesmo que exista esse ciclo.
2. Podemos usar o algoritmo de Dijkstra para encontrar os caminhos mais curtos em um gráfico com pesos negativos? Existe uma idéia: calcule o valor do peso mínimo, adicione um valor positivo (igual ao valor absoluto do valor do peso mínimo) a todos os pesos e execute o algoritmo Dijkstra para o gráfico modificado. Esse algoritmo funcionará?

Implementação simples do algoritmo Bellman-Ford

Fontes:

www.youtube.com/watch?v=Ttezuzs39nk

en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm

www.cs.arizona.edu/classes/cs445/spring07/ShortestPath2.prn.pdf

More articles:

- Slurm SRE - aprenda a garantir a felicidade do usuário
- Casa inteligente com Xiaomi no exemplo de uma sauna
- Viagem ao centro ... imagem do docker. Ou como baixar uma imagem do registro sem janela de encaixe

- Samsung Moscow Center for Artificial Intelligence nas histórias de funcionários
- SwayWM - UnixPorn você mesmo
- Meetup sobre AWS_Ru em Raiffeisenbank
- VVVVVV ??? VVVVVV !!! :)
- Glean Insights sobre as 18 melhores estruturas Java a serem usadas em 2020
- Convulsões
- Sete missões espaciais mais emocionantes do próximo ano

All Articles

Weekly-Geekly ES | 2021 Contact Us