Ata da questão 24.3–7

Alexandre Tachard Passos João Meidanis

17 de junho de 2010

1 Enunciado

A questão pede que modifiquemos o algoritmo de Dijkstra pra rodar em $O((V + E) \lg W)$ dado que todos os pesos das arestas do grafo estão contidos no intervalo $0 \ldots W$.

2 Solução

Em primeiro lugar, vamos modificar o algoritmo de Dijkstra descrito no livro para só colocar na estrutura de mínimo vértices que tenham $d[v]$ menor que infinito. Com isto, o algoritmo pára quando a estrutura de mínimo ficar vazia. Os vértices restantes não são alcancáveis a partir da origem e ficarão com seu $d[v]$ igual a infinito. Outra modificação diz respeito ao RELAX: ao invés de sempre atualizar com DECREASE-KEY, é necessário inserir um novo vértice na estrutura de mínimo quando sua distância anterior era infinito.

Desta forma, observamos que, a cada momento do algoritmo modificado de Dijkstra, todas as estimativas de distâncias $d[v]$ para vértices na estrutura de mínimo estão contidas no intervalo $[m, m + W]$, onde m e' o valor da menor chave na estrutura. Isto pode ser provado por indução da seguinte forma:

- Caso base: a estrutura é inicializada apenas com o vértice origem s com chave 0, logo os valores estão contidos em $[0, W]$.
- Passo indutivo: suponha que no início de uma certa iteração os valores das chaves estão contidos em $[m, m + W]$, onde m é a chave mínima. A operação EXTRACT-MIN pegará um vértice u de chave m. Sua expansão aos vizinhos dará origem a atualizações de chaves e inserções de novas chaves, mas todos os novos valores cairão no intervalo $[m, m + W]$, pois partirão de u e as arestas têm peso no máximo W. Não há arestas negativas, portanto não há o risco de aparecer uma chave nova menor que m. Assim, permanecemos num intervalo de tamanho no máximo $W + 1$ até o final da iteração. □

Assim, um heap de tamanho $W + 1$ será suficiente para guardar todas as distâncias parciais do algoritmo modificado de Dijkstra. Vamos agora implementar as operações EXTRACT-MIN, RELAX e INSERT em tempo hábil.

Para este algortimo, usaremos as seguintes estruturas:

1. H: um heap mínimo binário de tamanho $W+1$ para guardar as distâncias dos vértices. Cada elemento dessa heap é um inteiro.

2. P: um vetor de tamanho $W+1$, indexado pelos inteiros $0, \ldots, W$, para guardar listas duplamente ligadas de vértices que terão a mesma estimativa de distância $d[v]$. Na posição m do vetor ficarão os vértices com $d[v] \bmod (W+1) = m$. Como nunca haverá na estrutura dois vértices com estimativas que difiram em mais do que W, não haverá conflitos.

Na inicialização, todos os elementos de P recebem listas vazias e H está vazio, como na rotina INIT (Algoritmo 1). A complexidade é $O(W)$.

Algorithm 1 INIT

```
1: for i ← 0 to W do
2:     P[i] ← vazia
3: end for
4: H ← vazio
```

3 Insert

O próximo passo é inserir a origem na estrutura. Posteriormente, será necessário inserir outros vértices. Para tanto, o mesmo procedimento INSERT será usado. O código aparece no Algoritmo 2. Observe que a estimativa de distância d será inserida no heap apenas se esta estimativa é nova, ou seja, nunca apareceu um vértice com tal estimativa antes. Sabemos disto olhando a lista duplamente ligada na posição $m = d \bmod (W+1)$. Se ela for vazia, d é nova; caso contrário, já está no heap e não é preciso inseri-la novamente. Porém, em qualquer caso, o vértice v é colocado na lista duplamente ligada correspondente.

Não detalharemos o código de LISTA-INSERT. Trata-se de uma inserção em uma lista duplamente ligada que ocorre em $O(1)$. A complexidade de pior caso do INSERT como um todo é $O(\lg W)$, por causa da inserção no heap.

Algorithm 2 INSERT(v, d)

```
1: m ← d mod (W + 1)
2: if P[m] é vazia then
3:     LISTA-INSERT(P[m], v)
4:     HEAP-INSERT(H, d)
5: else
6:     LISTA-INSERT(P[m], v)
7: end if
```

4 Extract-min

Para EXTRACT-MIN, temos que obter a estimativa de distância mínima d no heap, e, a seguir, um vértice u com $d[u] = d$. Este vértice será um qualquer da lista $P[d \bmod (W+1)]$ (pegamos o primeiro). O problema é que esta lista pode ser vazia, pois, ao atualizar uma estimativa, não nos preocupamos em remover a estimativa anterior do heap quando sua lista fica vazia. Daria muita dor de cabeça. Preferimos consultar repetidamente o heap até conseguir uma lista não vazia, que necessariamente existe pois o INSERT garante que, para cada lista não vazia em P há o correspondente valor em H. Veja o código na Algoritmo 3.

Algorithm 3 EXTRACT-MIN

```
1: m ← MINIMUM(H) mod (W + 1)
2: while P[m] é vazia do
3:     HEAP-EXTRACT-MIN(H)
4:     m ← MINIMUM(H) mod (W + 1)
5: end while
6: return EXTRACT-FIRST(P[m])
```

Uma preocupação surge aqui com a complexidade deste código. Se vamos repetidamente extrair mínimos até encontrar uma lista não vazia, isto fura a expectativa de ter um passo $O(\lg W)$. Porém, a análise agregada das chamadas de HEAP-EXTRACT-MIN nos dizem que acontecem no máximo $O(E)$ tais chamadas. Para se convencer disto, note que, para poder sair, tem que ter entrado. Logo o número total de vezes que chamamos HEAP-EXTRACT-MIN não pode ser superior ao número de vezes que chamamos HEAP-INSERT, sob pena de esvaziar o heap. Mas o heap não está vazio, senão teríamos interrompido o loop principal de Dijsktra.

Por sua vez, as chamadas a HEAP-INSERT são feitas ao inserir a origem s e dentro do RELAX. Como o RELAX ocorre no máximo E vezes, temos o nosso resultado.

Concluimos que, embora não possamos garantir que EXTRACT-MIN execute em $O(\lg W)$ toda vez, o gasto agregado com chamadas a HEAP-EXTRACT-MIN, que dominam o tempo de execução desta rotina, é $O(E \lg W)$.

5 Relax

Implementar RELAX é simples. A implementação pode ser vista no Algoritmo 4.

Como pode-se ver, no total gastamos $O(\lg W)$, por causa do INSERT. É preciso ter uma maneira imediata de ir do vértice ao nó na lista duplamente ligada em que ele está, assim como tivemos no exercício anterior (24.3–6).

Algorithm 4 RELAX(u, v, w)

```
1: if d[v] > d[u] + w(u, v) then
2:     if d[v] < ∞ then
3:         LISTA-DELETE(P[d[v] mod (W + 1)], v)
4:     end if
5:     d[v] ← d[u] + w(u, v)
6:     π[v] ← u
7:     INSERT(v, d[v])
8: end if
```

6 Conclusão e análise

O algoritmo de Dijkstra executa V operações EXTRACT-MIN e E operações RELAX. Com os custos mostrados nas seções anteriores, é claro que essa implementação vai ter custo total $O((V + E) \log W + W)$. A parcela de tamanho W será absorvida pelo resto se $W \leq V$. Caso W seja maior que V, a implementação vista em classe de Dijkstra com heap binário já dá $O((V+E) \lg W)$. Assim, sempre há uma forma de implementar com complexidade $O((V + E) \lg W)$.