Projeto de Algoritmo com Implementação nº 2

MC458 - 2s2020 - Tiago de Paula Alves - 187679

1 Definições

Um alpinista deve escalar uma parede representada por uma grade $n \times m$, em cada célula (i, j) da grade tem um custo $C_{i,j} > 0$ associado. O alpinista deve começar na base, linha 1, e chegar no topo, linha n, tentando minizar o grau de periculosidade (custo total) do caminho. Quando ele se encontra na célula (i, j), com $1 \leq i < n$ e $1 \leq j \leq m$, os únicos movimentos possíveis são para as células $(i+1, j-1)$, $(i+1, j)$ ou $(i+1, j+1)$, quando elas fazem parte da parede.

1.1 Definições Adicionais

Caminho até o Topo: Considere um caminho $P = (p_1, \ldots, p_k)$, sendo k a quantidade de vértices no caminho. Então, P é um *caminho até o topo* se ele termina em alguma célula do topo da parede, ou seja, $p_k = (n, j)$ para algum $1 \leq j \leq m$.

Risco: O grau de periculosidade ou risco $R(P)$ de um caminho $P = (p_1, \ldots, p_k)$ é dado pela soma dos custos de cada célula do caminho, isto é,

$$R(P) = \sum_{1 \leq i \leq k} C_{p_i}$$

Risco Mínimo: O risco mínimo $R^*_{i,j}$ de uma célula (i, j) é o risco de um *caminho ótimo* de (i, j), ou seja, um caminho até o topo P com menor custo total $R(P)$ dentre os caminhos partindo de (i, j).

2 Solução

Teorema (subestrutura ótima)**.** *Seja $k \geq 2$ e suponha um caminho $P = (p_1, p_2, \ldots, p_k)$ até o topo partindo de p_1 e com risco mínimo $R(P) = R^*_{p_1}$. Então, o subcaminho $(p_2, \ldots, p_k)$ tem o menor risco possível partindo de p_2.*

Demonstração. Seja $S = (p_2, \ldots, p_k)$ o subcaminho de P mencionado e suponha um caminho até o topo $S' = (s_1, \ldots, s_l)$ partindo de $s_1 = p_2$, com risco $R(S') = R^*_{p_2}$ mínimo. Assim, temos o caminho $P' = (p_1, s_1, \ldots, s_l)$, com risco

$$R(P') = C_{p_1} + \sum_{i=1}^{l} C_{s_i} = C_{p_1} + R(S')$$

Suponha, por contradição, que $R(S') < R(S)$. Então, temos que

$$\begin{aligned} R(P') &= C_{p_1} + R(S') \\ &< C_{p_1} + R(S) \\ &= C_{p_1} + \sum_{i=2}^{k} C_{p_i} \\ &= R(P) \end{aligned}$$

Logo, P não é o caminho partindo de p_1 com menor risco, o que contradiz a suposição inicial.

Portanto, $R(S) \leq R(S') = R^*_{p_2}$, isto é, S tem risco mínimo dentre os caminhos partindo de p_2. ∎

2.1 Fórmula de Recorrência

Podemos notar que o caminho das células do topo com risco mínimo devem conter apenas a própria célula, então, $R^*_{n,j} = C_{n,j}$ para $1 \leq j \leq m$. Agora, para as células que não estão no topo, vamos ter que qualquer

caminho até o topo, inclusive um caminho ótimo $P = (p_1, \ldots, p_k)$, terá no mínimo 2 vértices, ou seja, $k \geq 2$. Portanto, pela subestrutura ótima, temos que $R^*_{p_1} = C_{p_1} + R^*_{p_2}$.

Como p_1 deve ser uma célula válida, então temos $1 \leq i < n$ e $1 \leq j \leq m$ tal que $p_1 = (i, j)$. No entanto, o alpinista só pode ir de (i, j) para as células superiores esquerda, central ou direita, isto é, $p_2 = (i+1, j-1)$ ou $p_2 = (i+1, j)$ ou $p_2 = (i+1, j+1)$, dado que essas posições são válidas. No caso geral, em que $1 < j < m$, temos que

$$R^*_{i,j} = \min\left\{C_{i,j} + R^*_{i+1,j-1}, C_{i,j} + R^*_{i+1,j}, C_{i,j} + R^*_{i+1,j+1}\right\}$$

Quando $j - 1 < 1$ ou $j + 1 > n$, podemos expandir a definição de risco mínimo para que $R^*_{i,j} = \infty$, mantendo a válidade do caso geral. Assim, temos a seguinte relação recorrente:

$$\begin{aligned}
R^*_{n,j} &= C_{n,j} && \text{para } 1 \leq j \leq m \\
R^*_{i,j} &= \infty && \text{para } 1 \leq i \leq n \text{ e } j < 1 \text{ ou } j > m \\
R^*_{i,j} &= C_{i,j} + \min\left\{R^*_{i+1,j-1}, R^*_{i+1,j}, R^*_{i+1,j+1}\right\} && \text{para } 1 \leq i < n \text{ e } 1 \leq j \leq m
\end{aligned}$$

Note que cada linha i depende apenas da linha superior $i+1$, com exceção do caso base $i = n$, que não depende de subcaminhos. Então, podemos calcular toda a matriz de risco mínimo $R^*_{i,j}$ a partir do topo da parede, sem necessidade de recursão ou memorização.

O resultado final R^* do algoritmo é apenas menor dos riscos dentre os caminhos partindo da base, isto é

$$R^* = \min_{1 \leq j \leq m}\left\{R^*_{1,j}\right\}$$

3 Algoritmo

O algoritmo foi baseado em programação dinâmica, seguindo a recorrência apresentada com pequenas modificações. Para evitar tratamento das bordas da matriz R e acessos repetidos, as variáveis *R-esq*, *R-cen* e *R-esq* foram usadas como uma janela dos possíveis risco mínimos partindo de (i, j), como mostra a figura 1.

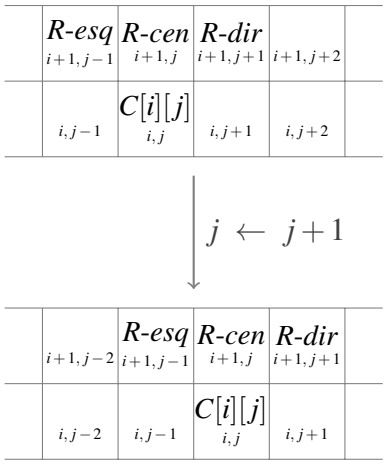

Figura 1: Janela de possíveis subcaminhos.

Desse modo, só é necessário um novo acesso para a variável *R-dir*. Na última posição da linha, no entanto, $(i+1, m+1)$ não é uma célula válida. Então, podemos usar a definição de risco mínimo expandida da seção Fórmula de Recorrência, tratando essa posição de forma especial.

Note que o algoritmo assume n e m positivos.

```
RISCO-MÍNIMO(C, n, m)
 1  Seja R[1..n][1..m] uma nova matriz.
 2
 3  para j = 1 até m
 4      R[n][j] ← C[n][j]
 5
 6  para i = n − 1 descendo até 1
 7      R-esq ← ∞
 8      R-cen ← ∞
 9      R-dir ← C[i][1]
10
11      para j = 1 até m − 1
12          R-esq ← R-cen
13          R-cen ← R-dir
14          R-dir ← R[i+1][j+1]
15
16          R-min ← min(R-esq, R-cen, R-dir)
17          R[i][j] ← C[i][j] + R-min
18
19      R-esq ← R-cen
20      R-cen ← R-dir
21      R-dir ← ∞
22
23      R-min ← min(R-esq, R-cen, R-dir)
24      R[i][m] ← C[i][m] + R-min
25
26  R-min ← R[1][1]
27  para j = 2 até m
28      R-min ← min(R-min, R[1][j])
29  retorna R-min
```

3.1 Implementação

O programa foi implementado em linguagem C, onde as matrizes C e R foram representadas em um buffer sequencial com ordem *row-major*. Nessa representação, os elementos de uma mesma linha são guardados sequencialmente de forma que as posições (i, j) e $(i, j+1)$ ficam próximas na memória, mas (i, j) e $(i+1, j)$ podem ficar distantes.

Além disso, pode-se notar dos casos de teste que $n \leq 100$, $m \leq 2200$ e $C_{i,j} \leq 99$. Então, o custo $C_{i,j}$ foi representado por um inteiro de 8 bits (`uint8_t`), já que sempre $C_{i,j} < 100 < 2^8$. O risco também pode ser limitado por

$$\begin{aligned} R_p^* &\leq \max_{1 \leq i \leq n \text{ e } 1 \leq j \leq m} \left\{ R_{i,j}^* \right\} \\ &= \max_{1 \leq j \leq m} \left\{ R_{1,j}^* \right\} \\ &\leq \sum_{i=1}^{n} \max_{1 \leq j \leq m} \left\{ C_{i,j} \right\} \\ &\leq \sum_{i=1}^{n} 99 \\ &= 99n \\ &\leq 9900 \\ &< 2^{16} \end{aligned}$$

Então, o risco foi representado com um inteiro de 16 bits (`uint16_t`). Por causa disso, o valor de infinito foi usado como `UINT16_MAX`, que é garantidamente maior ou igual a 2^{16}. De qualquer forma, o código fonte foi pensado de forma a facilitar a alteração dessas restrições, com as macros `CUSTO_WIDTH` e `RISCO_WIDTH`.

4 Complexidade

Podemos assumir que as operações aritméticas e as de acesso tem ordem $\Theta(1)$. Logo, o tempo de execução t_i de cada uma dessas operações i pode ser limitado pelas constantes $a \leq t_i \leq b$. Assim, o tempo do algortimo apresentado na seção anterior, para n e m positivos, pode ser majorado por

$$\begin{aligned} T(n,m) &\geq a + \sum_{j=1}^{m} a + \sum_{i=1}^{n-1} \left(a + \sum_{j=1}^{m-1} a \right) + \sum_{j=2}^{m} a \\ &= a + ma + (n-1)(1 + (m-1)a) + (m-1)a \\ &= nma + ma \\ &\geq a \cdot nm \end{aligned}$$

Da mesma forma, temos que

$$T(n,m) \;\leq\; b + \sum_{j=1}^{m} b + \sum_{i=1}^{n-1} \left(b + \sum_{j=1}^{m-1} b \right) + \sum_{j=2}^{m} b \;\leq\; 4b \cdot nm$$

Logo, a complexidade de tempo do algoritmo é $T(n,m) \in \Theta(nm)$.

Para o espaço, podemos ver que o único armazenamento adicional é da matriz R, com dimensões $n \times m$. Como a função não faz chamadas recursivas, temos que

$$E(n,m) = nm + \Theta(1) = \Theta(nm)$$

Assumindo que $m \in O(n)$, podemos afirmar então que a complexidade de tempo é $T(n) \in O\left(n^2\right)$ e de espaço também é $E(n) \in O\left(n^2\right)$.