Trabalho Prático 2: Eu, robô

Cleiton Neves Santos

May 31, 2016

1 Introdução

Dado um mapa da região, o modelo cinemático do robô e os pontos de partida e chegada o problema consiste em encontrar o caminho mínimo entre os dois pontos. O mapa, que é lido como um arquivo, pode ter três tipos de células:

- obstáculo: Impedem a passagem do robô;
- atalho: Permite que o robô se mova sem custo entre os atalhos do mapa;
- caminho: Células comuns de caminho, cada uma com seu respectivo peso.

Para resolução do problema será utilizado o algoritmo Dijkstra de caminho mínimo.

2 Solução do Problema

Como as restrições cinemáticas do robô variam a cada entrada, não foi implementado um tipo abstrato de grafos específico, pois isso implicaria percorrer todo o mapa e montar o grafo antes de executar o Dijkstra. Em vez disso foi implementada a função "Adjacentes" que, quando solicitado pela função Dijkstra encontra os adjacentes e seus respectivos pesos com relação ao nodo do qual foi solicitado.

input : matriz $G[N, M]$;(rx, ry)restrição de movinto ;(x, y) célula da qual se quer adjacentes;
output: vetor de adjacentes de (x, y)
begin
  if $(rx, ry) \neq 0$ **then**
    foreach *posição válida* p *em* G *dado* $p \leftarrow (x \pm rx, y \pm ry)$ **do**
      $adjacentes[cont] \leftarrow p$
      $cont \leftarrow cont + 1$
    end
  else
    foreach *posição válida* p *em* G *dado* $p \leftarrow (x \pm 1 \oplus y \pm 1)$ **do**
      $adjacentes[cont] \leftarrow p$
      $cont \leftarrow cont + 1$
    end
  end
end

Algorithm 1: Adjacentes

input : matriz $G[N, M]$, S célula origem;
output: menor caminho de S à todos os vértices
begin
  cria vetor de vertices Q
  foreach *célula* v *na matriz* G **do**
    $distancias[v] \leftarrow INFINITO$;
    $antecessor[v] \leftarrow invalido$;
    $Q \leftarrow v$
  end
  $distancias[S] \leftarrow 0$
  while Q *não vazio* **do**
    $u \leftarrow$ remove vertice com menor distancia em Q
    foreach *célula* v *adjacente a* u **do**
      $aux \leftarrow distancia[u] +$ distancia entre u e v;
      if $aux > distancia[v]$ **then**
        $distancia[v] \leftarrow aux$
        $anterior[v] \leftarrow u$
      end
    end
  end
end

Algorithm 2: Dijkstra

3 Análise de Complexidade

3.1 Adjacentes

A função $Adjacentes()$ explicitada código acima calcula os nodos adjacentes, como independente do tamanho da entrada cada vértice terá no máximo quatro adjacentes a função os calcula em $O(1)$ e também calcula o peso de cada adjacente com somas sucessivas do tamanho da restrição do modelo cinemático então $max(O(rx), O(ry))$ exceto no caso em que o mapa tem atalhos onde calcula em $O(n\_atalhos * max(rx, ry))$ porém isso só pode ocorrer uma vez, já que a lista de adjacentes dos atalhos é salva.

3.2 Dijkstra

Como pode ser visto no pseudocódigo acima um loop para cada célula da matriz que representa um vértice é feita duas atribuições com complexidade $O(1)$ inclusive a inclusão do vértice no vetor Q.Sendo assim, essa parte do algoritmo tem complexidade $O(V)$ sendo $V = N * M$ o número de vértices.

Como foi implementado com Heap Queue após a primeira parte do código é chamada a função "$Constroi()$" sobre o vetor Q que constroi um heap em $\theta(V * log(V))$ no pior caso, como é característico dessa estrutura. Até esse ponto a complexidade é dada por $O(V) + O(V * log(V))$ ou seja $O(V * log(V))$.

No primeiro *while* do pseudocódigo, segundo *loop*, se repete até que a fila Q esteja vazia, ou seja, V vezes. Todas elas retirando o vértice de menor distância da fila, a retirada é feita em $O(1)$ porém a mesma função também chama a operação para refazer o Heap $Refaz()$ que percorre toda a altura do mesmo para tal $O(log(V))$.

É também chamada a função $Adjacentes()$ da qual a complexidade já foi calculada. Depois ainda dentro do *loop* para cada adjacente é feito atualização de seu peso na fila que é no pior caso $O(log(V))$ e outras atribuições $O(1)$.
Sendo assim a complexidade no pior caso é $O(V * (ln(V) * max(O(rx), O(ry)) + 4 * ln(V)) \rightarrow O(V * ln(V) * (max(O(rx), O(ry)))$ que é também a complexidade do total algoritmo já que $O(x + y)$ =max$(O(x), O(y))$.

3.3 Espacial

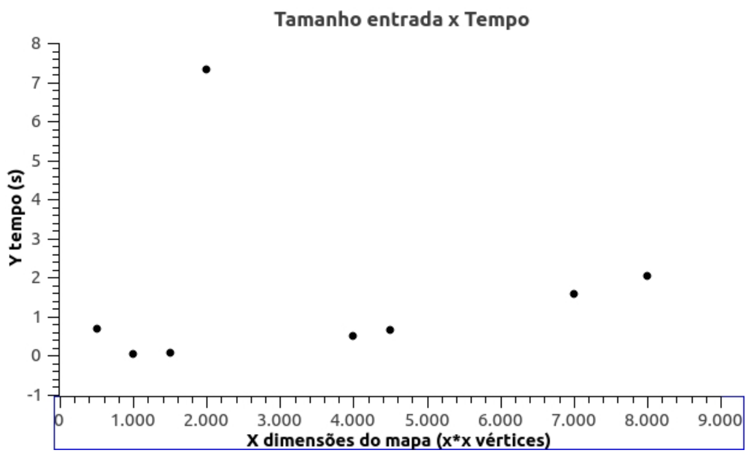
A complexidade de espaço fica em $O(N * M) = O(V)$ pois o tamanho da matriz alocada para representar o mapa e outros vetores cresce dessa forma de acordo com a entrada.

4 Análise Experimental

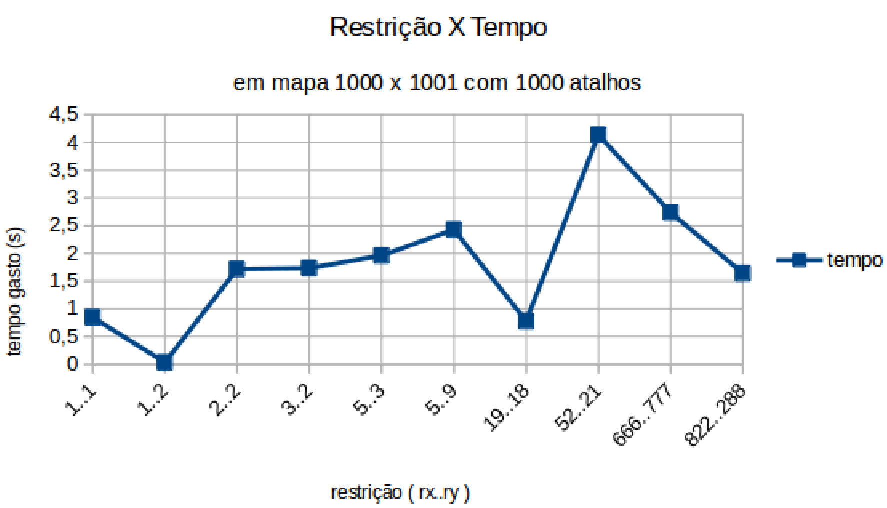
Para realizar os experimentos foi feito um gerador de matrizes que que tem como saída uma matriz com dimensões e número de atalhos e obstáculos desejados. Para medir o tempo de execução da função dijkstra foi utilizada a biblioteca "$< time.h >$" para contar o número de clocks necessários para execução da função. Dada a contagem do número de clocks, a conversão para segundos é simples visto que tal biblioteca fornece o número de clocks por segundo através da variável $CLOCKS\_PER\_SEC$.

Cada instância fornecida pelo gerador foi executada 5 vezes e o tempo de execução foi obtido retirando a média de tempo das 5 execuções.Os testes foram realizados em uma máquina com sistema operacional Ubuntu 16.04, processador Pentium Dual-Core 3.20GHz e com 3GB de memória ram.

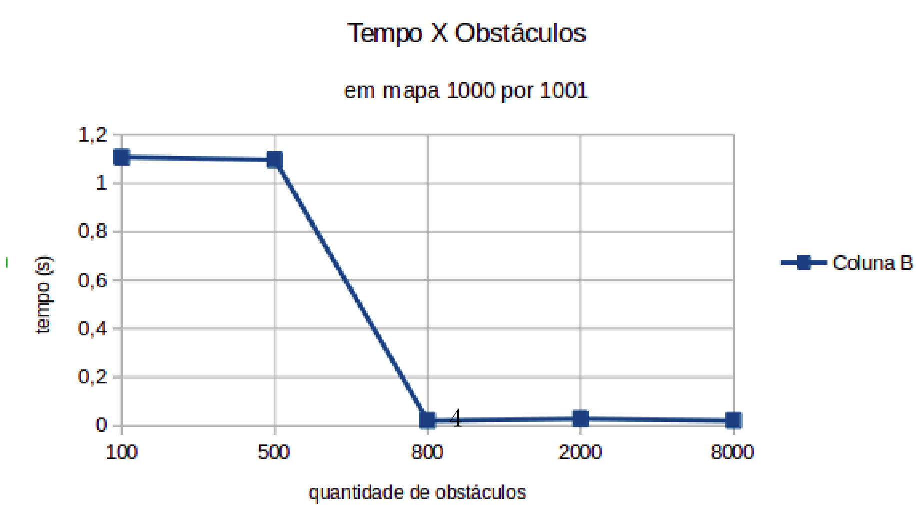
O primeiro gráfico a seguir exibe como o tempo de execução da função $Dijkstra()$ cresce em relação ao tamanho do mapa de entrada.



(a) tamanho x tempo



(b) atalhos e restrições



(c) quantidade de obstáculos e tempo

Figure 1: tabelas

Para os testes "*tamanho* x *tempo*" em um mapa sem atalhos foi executado o comando de encontrar o caminho mínimo com restrições rx=ry=0, inicio em x=y=0, e chegada em x=m-1, y =n-1. Os testes "*restrição* x *tempo*" seguem a mesma regra porém variando as restrições de movimento e o mapa possui atalhos.O eixo x representa a dimensão do lado do mapa, sendo assim o número de nodos igual a $x * x$.

Como pode ser observado no gráfico a seguir a restrição de movimento tem muito pouca influência no tempo de execução, Caso a implementação reduzisse o número de vértices com o aumento da restrição haveria uma redução no tempo de execução.

5 Conclusão

Neste trabalho foi resolvido o problema de encontrar o caminho mínimo dada uma restrição cinemática e um mapa com atalhos e obstáculos com uso o algoritmo Dijkstra otimizado com uma heap queue.

Tanto a complexidade em tempo quanto em espaço explicitam que seria uma melhor escolha implementar o grafo de forma que o aumento dos valores de restrição cinemática implicasse em uma diminuição do número de vértices.

6 Referências

- Projeto de Algoritmos - Nivio Ziviani (implementação do Heap_Min_Priority_Queue);
- Dijkstra's Algorithm - Wikipedia .
- What does 'Space Complexity' mean? - geeksforgeeks.org/g-fact-86/