TRABALHO PRÁTICO 2:
Problema do Banco Central

Bruno Maciel Peres

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

brunomperes@dcc.ufmg.br

Resumo. *Este relatório descreve duas soluções propostas para o problema do Banco Central, que é uma variação do problema da galeria de arte, ou (Art Gallery Problem, em inglês). Os principais objetivos deste trabalho são: (1) modelagem de um problema NP-Completo que envolve estruturas de grafos, (2) elaboração de diferentes heurísticas para solucionar um mesmo problema e (3) apresentar uma análise comparativa entre as abordagens propostas para solucionar o problema.*

1. INTRODUÇÃO

Neste trabalho, descrevemos e implementamos um algoritmo que utiliza duas heurísticas para resolver o problema do Banco Central. O AGP (*Art Gallery Problem*), como será referido a partir de então, é um problema conhecido e foco de estudo em Geometria Computacional, mais especificamente como um problema de visibilidade computacional. O problema foi proposto inicialmente em 1973 por Victor Klee, podendo ser conhecido também como Problema da Iluminação, além de possuir diversas variações, como guardas de posição variável, paredes ortogonais, possuir ou não obstáculos, entre outros.

O problema em questão consiste em encontrar o menor número de guardas para vigiar completamente um Banco (ou uma galeria de arte, no caso do problema original), representada por um polígono simples. Pela definição desta variante do problema, o Banco representado, não possui obstáculos (polígonos dentro deste polígono), as arestas representam as paredes e os vértices o encontro dessas paredes, os guardas só podem ser posicionados nos vértices e os ângulos entre as arestas não são necessariamente ortogonais.

Vigiar um vértice V_k é definido como posicionar um guarda em outro vértice V_i ou no próprio vértice V_k de forma que uma aresta entre V_k e V_i não atravesse outra aresta e esteja dentro do polígono. O ângulo de visão de um vértice é o ângulo interno do vértice.

Para solucionar o problema foram elaboradas duas heurísticas de abordagens diferentes, a primeira que parte da triangulação do polígono e execução de um algoritmo de 3-coloração e a segunda é baseada no problema de cobertura de vértices de um grafo.

Este problema é da classe NP-Difícil, como provado por Lee e Lin[5], logo as soluções propostas são algoritmos aproximativos para encontrar uma resposta próxima da solução ótima.

Foi realizado uma análise experimental do programa, executando sobre algumas entradas. As métricas para avaliar a execução do programa para cada polígono de quantidade de vértices v são: o tempo de execução e o resultado obtido, ou seja, a quantidade de guardas necessários para vigiar o Banco.

No algoritmo, os vértices são referenciados através de chaves ou ids, contadas de 1 a v e a quantidade de arestas será denotada por a. Essas nomenclaturas serão utilizadas para análise de complexidade de espaço e de execução dos algoritmos.

2. MODELAGEM

2.1. Estruturas de dados

O problema foi modelado a partir da estrutura de um grafo com lista de adjacências, onde os vértices do grafo são os vértices do polígono que representa o Banco e cada um desses vértices possui a lista de vértices adjacentes (possuem arestas com esse vértice). Utilizou-se alocação dinâmica de memória para armazenar as listas de adjacências, assim como para armazenar o vetor de vértices.

A implementação desse grafo foi baseada em uma listas duplamente encadeada adaptada às necessidades do problema, respeitando a modularização do programa. A adaptação do tipo lista partiu da implementação do Prof. Fernando V. Paulovich[4] e foi modificada para que o programa pudesse acessar e operar sobre esse novo Tipo Abstrato de forma eficiente.

Os vértices que possuem aresta visível a um outro vértice, têm sua id armazenada nos nós da lista de adjacência desse vértice.

Para a alocação do vetor e das listas são utilizados parâmetros contidos no arquivo de entrada antes da execução das heurísticas. São eles: (a) a quantidade de vértices contidos no polígono e (b) as coordenadas de cada vértice. As coordenadas são representadas como divisão de inteiros no arquivo de entrada, o que levou à decisão delas serem armazenadas como pontos flutuantes (tipo *float*).

Uma decisão tomada para modelar este problema foi que a lista de vértices visíveis a um outro vértice não pode conter redundância, visto que para a 2ª heurística a quantidade de vértices visíveis (tamanho da lista de adjacências) é decisória para a execução do algoritmo. O custo de verificar se um vértice já existe em uma lista é $O(v)$ para cada inserção na lista.

A complexidade de espaço deste programa para armazenamento do polígono, com todas suas arestas é $O(v^2)$, já que no pior caso de espaço deverão ser armazenados todos os outros vértices de um mesmo vértice (polígono convexo).

Abaixo a descrição dos Tipo Abstratos de Dados utilizados e seus elementos.

Para melhor legibilidade do código, foi-se decidido que o nome dos novos tipos criados serão em caixa alta, e nos argumentos das funções, os tipos mais abrangentes vêm antes dos mais específicos.

2.1.1. Polígono

Armazena informações de um polígono através da estrutura de um grafo.

1: GRAFO
Vetor de vértices;
Número de vértices;

2.1.2. Vértice

Armazena informações de cada vértice do polígono.

2: VÉRTICE
lista das ids dos vértices adjacentes;
Coordenada do vértice;
Flag de status se possui guarda;
Cor do vértice;
id vértice anterior no polígono;
id vértice seguinte no polígono;

A cor do vértice é utilizada durante o algoritmo de 3-coloração, na heurística 1 e para indicar status de coberto ou não na heurística 2.

3. SOLUÇÃO PROPOSTA

As soluções propostas para o problema diferem completamente, logo serão abordadas individualmente a seguir, mas ambas possuem complexidade assintótica $O(v^3)$.

Assim, a complexidade de execução deste programa é $O(i \times v^3)$, onde i é a quantidade de instâncias executadas.

Por se tratar de um problema NP-Difícil, foi-se necessário a utilização de algoritmos aproximativos. Com relação à um limite inferior para cálculo da taxa de aproximação foi-se mostrado que esse problema é APX-Difícil, usando reduções de conservação de intervalo do problema 5-OCCURRENCE-MAX-3-SAT. Isso quer dizer que, uma constante $\varepsilon > 0$ tal que a taxa de aproximação $1 + \varepsilon$ não pode ser garantida por nenhum algoritmo aproximativo em tempo polinomial a menos que $P = NP$ [11].

3.1. Heurística 1

A heurística 1 aborda o problema através da triangulação do polígono que resulta em $n-2$ triângulos, e em seguida, executa-se a 3-coloração do grafo gerado (polígono triangularizado) que sempre é possível para qualquer polígono simples triangularizado [6].

Após o procedimento de coloração do grafo, defini-se qual a cor menos utilizada e posiciona-se um guarda em cada um dos vértices que possuem essa cor. Assim, a quantidade de vértices com guardas será sempre $\lfloor n/3 \rfloor$ [6].

A complexidade de assintótica de execução dessa heurística é $O(v^3)$, pelo custo de calcular visibilidade de um vértice genérico em $O(v^2)$ para todos os vértices com guarda, que é uma fração de v, $\lfloor v/3 \rfloor$. Multiplicados a fração e o custo da visibilidade chegamos à complexidade cúbica.

3.1.1. Triangulação

A complexidade assintótica dessa função é $O(v^2)$, visto que deve-se percorrer todos os vértices $O(v)$, e ao inserir um vértice na lista de adjacência de outro vértice, verifica-se se não há redundância $O(v)$.

O método de triangulação utilizado é o de corte de orelhas, que consiste em verificar se existe algum outro ponto dentro de um possível triângulo e verificar se os 3 pontos estão no sentido anti-horário (garante que a aresta formada estará dentro do triângulo). Caso as condições sejam satisfeitas, a aresta é adicionada entre dois pontos, formando o triângulo. Para mais detalhes, vide [7].

3: triangulação()

```
toma-se todos os vértices i, vértices à direita de i, r[i] e à esquerda de i, l[i] de cada vértice;
while quantidade de triângulos é < (v − 2) do
    if existe orelha entre (l[i], i, r[i]) then
        if caso não exista aresta entre l[i] e r[i] then
            insere aresta mutuamente entre l[i] e r[i];
        end
        insere l[i], i, r[i] num novo triângulo no vetor de triângulos;
        l[r[i]] = l[i];
        r[l[i]] = r[i];
        quantidade de triângulos++;
    end
    return vetor de triângulos formados;
end
```

3.1.2. 3-Coloração

A complexidade assintótica dessa função é $O(t^2)$, onde t é a quantidade de triângulos no polígono, pois a busca por triângulos com apenas dois vértices coloridos é $O(t)$ e esse procedimento acontece para todos v vértices.

Pela definição do procedimento de triangulação, $t = n - 2$. Logo a complexidade dessa coloração é $O(v^2)$.

4: 3-coloração(triângulos gerados na triangulação)

```
colore os vértices do primeiro triângulo no vetor, cada um com uma cor;
while Enquanto houverem vértices para colorir do
    busca o triângulo que possui apenas dois vértices coloridos;
    colore esse triângulo com a cor ainda não utilizada nesse triângulo;
    aumenta contador da cor utilizada em 1;
end
return cor menos utilizada;
```

3.2. Heurística 2

A heurística 2 busca uma solução para o problema através de um algoritmo de busca gulosa, buscando solucionar a cobertura de vértices tomando o vértice que cobre mais vértices naquela iteração e removendo os vértices já cobertos das listas de todos os vértices.

Nível de aproximação: $O(logn)$, por aproximação do problema com o problema de cobertura de conjuntos [11].

A complexidade de execução final dessa heurística é $O(v^3)$, devido ao cálculo da visibilidade de todos os vértices v em $O(v^2)$ cada, remoção dos vértices já cobertos em

$O(v^2)$ para cada posicionamento de um novo guarda e recálculo da visibilidade para os vértices que possuem um guarda.

3.2.1. Cobertura de vértices

5: cobertura(Grafo G)

```
Calcula a visibilidade de todos os vértices;
while Há vértices não cobertos do
    Busca o vértice que cobre mais vértices;
    Posiciona um guarda neste vértice;
    Apaga os vértices cobertos por esse guarda da lista de todos os outros vértices;
end
for i=1; i ≤ v; i++ do
    if vértice i possui guarda then
        recalcula a visibilidade do vértice i;
    end
end
```

4. IMPLEMENTAÇÃO

O critério de separação dos arquivos do programa objetivou a modularização do programa, para possível posterior utilização, evitar duplicamento de funções, além de melhorar a legibilidade e organização do mesmo.

4.1. Código

4.1.1. Arquivos .c

- **main.c:** Arquivo principal do programa, contém as chamadas de funções de entrada e saída e a chamada de execução das heurísticas.
- **io.c:** Realiza a comunicação do programa com o ambiente, lendo o arquivo de entrada e armazenamento das informações lidas e escrevendo os resultados.
- **heuristica1.c:** Define o funcionamento da heurística 1.
- **heuristica2.c:** Define o funcionamento da heurística 2.
- **lista.c:** Define as funções que operam sobre o TAD lista criado.
- **grafo.c:** Define as funções que operam sobre o TAD grafo.
- **poligono.c:** Contém funções comuns à operações sobre polígonos.
- **ordena.c:** Contém funções para ordenação de vetor necessárias para impressão dos vértices cobertos por outro vértice ordenados crescentemente.
- **timerlinux.c:** Contém funções para medir o tempo de execução das heurísticas.

4.1.2. Arquivos .h

- **io.h:** Contém o cabeçalho das funções de entrada e saída.
- **heuristica1.h:** Define o cabeçalho da função de execução da heurística 1.
- **heuristica2.h:** Define o cabeçalho da função de execução da heurística 2.
- **grafo.h:** Define a estrutura do TAD e as funções que operam sobre o TAD grafo.
- **lista.h:** Define a estrutura do TAD lista e o cabeçalho de suas funções.

- **poligono.h:** Contém o cabeçalho das funções para operações comuns em polígonos e define estruturas básicas, como ponto, linha e triângulo.
- **ordena.h:** Contém o cabeçalho da função de ordenação necessária para ordenação de vetor.
- **timerlinux.h:** Cabeçalho e instruções para cronometrar o tempo de execução das heurísticas.

4.2. Compilação e execução

O programa deve ser compilado através do compilador GCC através do *makefile* anexo ou através do seguinte comando em terminal:

```
 gcc main.c lista.c grafo.c io.c heuristica1.c heuristica2.c
timerlinux.c poligono.c ordena.c -o tp2
```

Para execução do programa, são requeridos dois parâmetros, o nome do arquivo de entrada e do arquivo de saída, em qualquer ordem. Caso não haja ao menos 2 argumentos, o programa encerra a execução. A análise de execução do programa pode ser habilitada passando $-a$, o que habilita a escrita das métricas de avaliação do programa num arquivo de texto. O tamanho máximo de nome de arquivo é 255 caracteres. Um exemplo é dado a seguir.

```
 ./tp2 -i input.txt -o output.txt
```

4.3. Entrada e saída

4.3.1. Formato da entrada

O arquivo de entrada cujo nome é passado como argumento para o executável deve conter, na primeira linha, a quantidade de instâncias que este arquivo possui, ou seja, quantos polígonos diferentes estão contidos nesse arquivo e que deverão ser executadas pelo programa.

No início de cada instância, há o número v de vértices, e em seguida as coordenadas x e y de cada vértice, representadas através de divisão de inteiros. Os vértices devem ser apresentados no sentido anti-horário para o funcionamento do programa.

Um exemplo de arquivo de entrada é dado a seguir:

```
1
4
1/1 1/1
100/2 1/1
500/10 50/1
1/1 100/2
```

4.3.2. Formato da saída

A saída do programa contém informações sobre a alocação final dos guardas para cada heurística. Em cada linha há a id do vértice em que está o guarda e em seguida as ids dos vértices vigiados por este guarda ordenadas crescentemente.

Um exemplo de saída é dado abaixo:

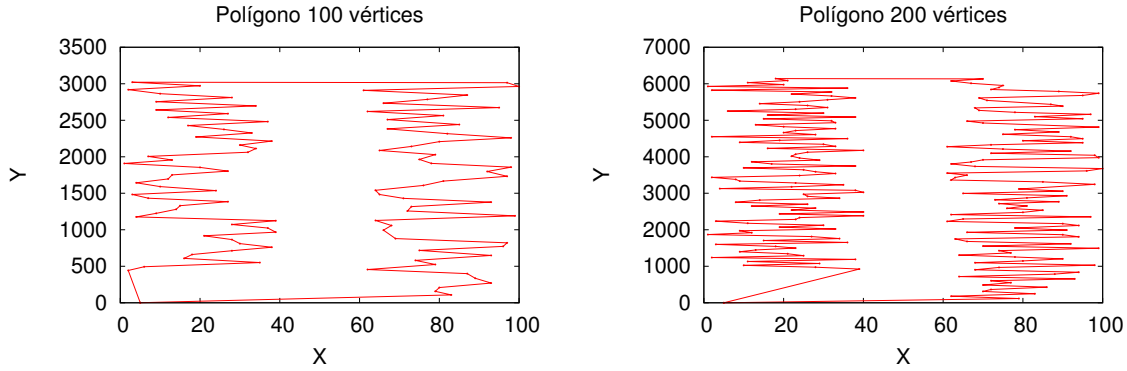
```
#Heuristica1:
1
2 : 1 2 3 4
#Heuristica2:
2
1 : 1 2 3 4
2 : 1 2 3 4
```

5. ANÁLISE EXPERIMENTAL

5.1. Gerador de entradas

O algoritmo criado foi executado sobre as entradas fornecidas (entradas com 12 vértices ou menos). Para analisar o algoritmo com entradas maiores, foi escrito um programa que gera um polígono construindo metade dos vértices de um lado do polígono, e em seguida a outra metade dos vértices é gerado. Esse gerador produz entradas inválidas acima do limite utilizado por ele, possivelmente por haver maior probabilidade dos pontos gerados serem colineares em entradas muito altas, mesmo as coordenadas sendo geradas aleatoriamente. O gerador está anexo aos arquivos deste trabalho.

Para minimizar casos desconsiderados pela especificação do trabalho, o gerador avalia se o ponto gerado não é colinear com o 2 pontos anteriores, através da inclinação dos segmentos de reta formados entre eles e se as coordenadas do ponto são iguais às do ponto anterior. Um exemplo de polígono gerado é dado abaixo.



5.2. Testes

Os testes foram realizados em uma máquina com processador Core i3 370M 2.40GHz, 4GB RAM DDR3 1333MHz e sistema operacional Linux Ubuntu 32 bits versão do kernel 3.2.0-24.

#Vértices	Tempo execução heurística 1 (seg.)	Tempo execução heurística 2 (seg.)	# Guardas heurística 1	# Guardas heurística 2
4	0.000013	0.000021	1	1
7	0.000048	0.000116	2	2
11	0.000079	0.000275	3	3
11	0.000116	0.000374	3	3
12	0.000042	0.000194	2	2
20	0.000124	0.000626	5	6
30	0.000707	0.002094	8	3
40	0.001044	0.005080	11	5
50	0.003210	0.015889	14	7
80	0.008459	0.022718	22	8
100	0.010814	0.042062	30	11
120	0.018804	0.059589	37	12
150	0.028055	0.102094	41	14
200	0.066763	0.239146	58	21
250	0.131800	0.397103	78	23

Tabela 1. Resultados testes

6. CONCLUSÃO

A partir da análise experimental deste programa, conclui-se que a estratégia gulosa utilizada na heurística 2 minimiza a quantidade de guardas, contudo o tempo de execução desse método é superior ao da heurística 1.

O trabalho atingiu seus principais objetivos: o estudo de problemas complexos, elaboração de dois métodos diferentes para solucionar o problema e análise das soluções propostas.

Dentre as adversidades encontradas durante o desenvolvimento desse programa, a maior delas foi o desconhecimento na área de Geometria Computacional, implicando em uma busca sobre as formas de modelar a estrutura para armazenar o polígono e operar sobre ele.

Algumas melhorias que poderiam ser consideradas neste trabalho são:

- Otimização da função de calcular a visibilidade de todos os vértices, para evitar que uma aresta previamente calculada seja testada novamente.
- Melhorar o gerador para que seja capaz de gerar entradas maiores que as utilizadas.

7. Referências

[1] Cormen, T. (2001). *Introduction to algorithms*. MIT press.

[2] Ziviani, N. (2007). *Projeto de Algoritmos com implementações em Pascal e C*. Pioneira Thomson Learning.

[3] Robert Sedgewick. *Algorithms in C*. Addison Wesley Professional, 1st edition, 2001.

[4] Paulovich, Fernando V. *Listas Duplamente Ligadas*. www.lcad.icmc.usp.br/∼paulovic/aulas/ED-I/SCC0202-aula-05-Listas_duplamente_ligadas.pdf

[5] D. T. Lee and A. K. Lin. *Computational complexity of art gallery problems*. IEEE Transactions on Information Theory, ISSN 0018-9448, 1986

[6] Fisk's Proof www.sciencedirect.com/science/article/pii/009589567890059X

[7] Steven S. Skiena, Miguel A. (2003) Revilla. *Programming challenges : the programming contest training manual*.

[8] Couto, Marcelo Castilho. *Um algoritmo exato para um problema de galeria de arte*. Campinas, 2010.

[9] Ghosh, S. K. (1987), *Approximation algorithms for art gallery problems*, Proc. Canadian Information Processing Society Congress.

[10] Marcelo C. Couto and Pedro J. de Rezende and Cid C. de Souza, (2009). www.ic.unicamp.br/∼cid/Problem-instances/Art-Gallery

[11] Ghosh, S. K. School of Technology & Computer Science. Mumbai, India. Approximation Algorithms for Art Gallery Problems in Polygons and Terrains www.tcs.tifr.res.in/∼ghosh/lec-psgtech-sept11.pdf