Jose Diego - 1761, Vitor - 3049

Trabalho de Projeto e Análise de Algoritmos

Brasil

25 de Outubro de 2018

Jose Diego - 1761, Vitor - 3049

Trabalho de Projeto e Análise de Algoritmos

Trabalho referente a implementação de algoritmos para resolução de problemas baseado em técnicas de programação Dinâmica estudadas em sala, para a disciplina de Projeto e Analise de Algoritmos.

Universidade Federal de Viçosa

Campos Florestal

Ciência da Computação

Brasil

25 de Outubro de 2018

Resumo

Em ciência da computação vários são os problemas enfrentados na implantação de uma solução baseado em um contexto da vida real, abstrair uma solução viável é sempre um desafio. Técnicas como programação Dinâmica vem ajudar os profissionais nessas áreas neste momento fornecendo meios pelo quais problemas podem ser divididos em sub-problemas parciais, e estes podem serem resolvidos e comporem a solução otima final, isto tudo de uma maneira mais eficiente como sera descrito neste texto. .

Palavras-chaves: algoritmos, programação Dinâmica, piramide.

Lista de ilustrações

Figura 1 – Ilustração da piramide 7
Figura 2 – Ilustração da piramide na matriz do programa 8
Figura 3 – Formatado do arquivo de entrada 8
Figura 4 – Imprementação da Função do recursivo Padrao 8
Figura 5 – Imprementação da Função do Memoization 9
Figura 6 – Imprementação da Função ultilizando a técnica deTrazPraFrente, por programação dinâmica. 9
Figura 7 – Menu do programa da Tarefa A. 10
Figura 8 – Resultado da opção 2) Imprimir Pirâmide. 10
Figura 9 – Resultado do algoritmo Recursivo Padrão. 10
Figura 10 –Resultado do algoritmo Memoization. 11
Figura 11 –Resultado do algoritmo, de trás pra frente. 11
Figura 12 –Rota para o melhor resultado. 11
Figura 13 –Resultados dos algoritmos implementados. 12
Figura 14 –Resultados dos algoritmos implementados. 12
Figura 15 –Imprementação do count 14
Figura 16 –Imprementação do clone 14
Figura 17 –Imprementação do Caminho 15
Figura 18 –Menu principal 15
Figura 19 –Menu principal 16
Figura 20 –Resuldtados Debug 17
Figura 21 –Tabela de tamanhos por tempo 17
Figura 22 –Grafico de tamanhos por tempo 18

Sumário

1 Introdução **5**

2 Metodologia **6**

3 Desenvolvimento **7**

3.1 Tarefa A 7

3.1.1 Descrição da Tarefa A 7

3.1.2 Implementação 7

3.1.2.1 Recursivo Padrão 8

3.1.2.2 Memoization 9

3.1.2.3 De tráz pra frente - Programação Dinâmica 9

3.1.3 Execução e Resultados 10

3.1.3.1 Comparando Resultados 11

3.2 Tarefa B 13

3.2.1 Descrição da Tarefa B 13

3.2.2 Implementação 13

3.2.3 Execução e Resultados 15

3.2.4 Comparando resultados 16

Conclusão **19**

Referências **20**

1 Introdução

O texto aqui apresentado refere-se o relatório do trabalho prático 3 da disciplina de programação e análise de algoritmos, neste trabalho será realizado a implementação de duas tarefas. Na primeira, Tarefa A, teremos as seguintes técnicas analisadas Recursivo Padrão, “Memoization ” e programação Dinâmica (“de trás pra frente”). Já na Tarefa B, será implementado o algoritmo da “Cidade com Quadriculado”.

No desenvolvimento do trabalho foi utilizado um máquina com sistema Ubuntu 16.04 64 bits, 6 GB de RAM e processador i5-3337U CPU 1.80 GHz.

Este trabalho segue dividido na seção 2 Metodologia, seguido da seção 3 Desenvolvimento detalhando a realização do trabalho e por fim a Conclusão e Referências.

2 Metodologia

A metodologia adotada consiste primeiramente, no estudo teórico da técnica de programação Dinâmica , para a solução de problemas, em seguida a implementação destes na linguagem C conforme a descrição das tarefas contidas no relatório do trabalho prático. Sobre a arquitetura utilizada como descrito na Introdução, foi utilizado a IDE Code Blocks para execução dos programas desenvolvidos, ferramenta escolhida devido ao seu grande uso por parte da comunidade da área e recomendações ao longo do curso.

Além disso foi utilizado a ferramenta de edição online overleaf para a criação e edição compartilhada deste texto. E o google drive para gerenciamento de versões do programa desenvolvido, tudo seguindo boas práticas de estruturas de dados como em Ziviani (2010).

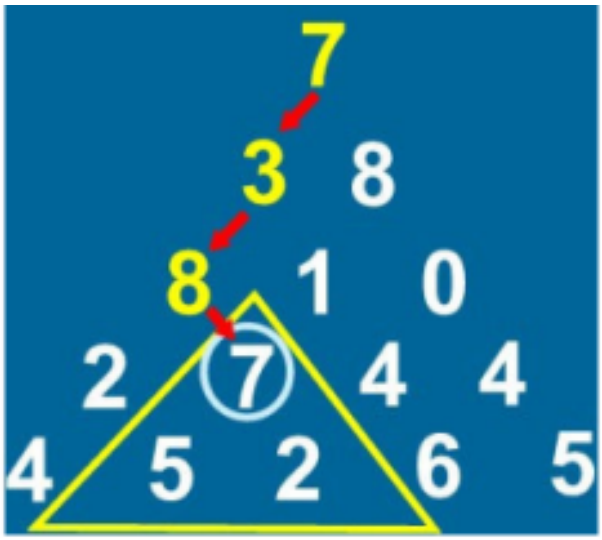
3 Desenvolvimento

3.1 Tarefa A

3.1.1 Descrição da Tarefa A

A tarefa A envolve o problema já visto em sala de aula, que é o problema de percorrer uma piramide de numeros cujo caminho é o de maior soma, na figura 1 podemos ver uma ilustração de como seria esta piramide.

Figura 1: Ilustração da piramide



Esta rota pelo qual se deve realizar a soma dos números deve ser realizada partindo do topo em direção a base como mostrado na ilustração.

3.1.2 Implementação

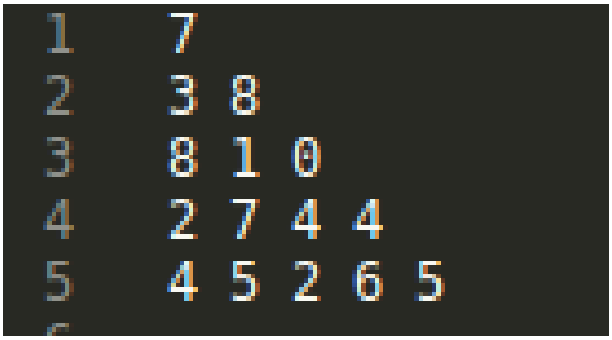
Na implementação do algoritmo podemos encontrar um TAD pirâmide , contendo duas matrizes uma para pirâmide original e outra para as soluções parciais, além de outras variáveis essenciais para guarda o tamanho da pirâmide e o modo debug. Apartir da figura 2 podemos ver na imagem uma representação da piramide pela matriz, com os repectivos dados.

A matriz é preenchida por um arquivo de entrada com o seguinte formato da Figura 3.

Figura 2: Ilustração da piramide na matriz do programa

	1	2	3	4	5
1	7				
2	3	8			
3	8	1	0		
4	2	7	4	4	
5	4	5	2	6	5

Figura 3: Formatado do arquivo de entrada



3.1.2.1 Recursivo Padrão

A técnica recursiva padrão traz uma grande facilidade de implementação, mas como já foi visto em sala de aula traz problemas de alto consumo de memória o que pode vir ocasionar estouro da pilha, além de realizar retrabalho, tornando o método restritivo. Na função temos a chamada recursiva até que a última linha da matriz seja atingida, ao final ela retorna o valor relativo a base da matriz e analisa qual o maior valor deste para a próxima posição j+1, logo depois este é somado ao valor acima, e de forma recursiva o processo ocorre até chegar ao topo, retornando assim o maior valor entre a base e o topo.

Figura 4: Imprementação da Função do recursivo Padrao

```
329  int recursivoPadrao(int i, int j, Piramide p){
330      if(i == p->tamanho-1)
331      {
332          return p->matriz[i][j];
333      }
334      else
335      {
336          return (p->matriz[i][j] + maximo(recursivoPadrao(i+1, j, p), recursivoPadrao(i+1, j+1, p)));
337      }
338  }
```

Apartir da figura 22 podemos ver um trecho de codigo da tecnica apresentada no trabalho proposto.

3.1.2.2 Memoization

Ainda utilizando recursividade como no na item anterior, mas agora juntamente armazenando em uma tabela os valores anteriormente calculados. Há a possibilidade então de recuperar estes respectivos valores, assim desta maneira provocando uma economia de tempo e espaço. Isto pode ser visto através da tabela[i][j] na Figura 5 onde ela fica responsável por armazenar o resultados dos subproblemas ótimos. Onde ao fim temos a solução final do algoritmo.

Figura 5: Imprementação da Função do Memoization

```
353
354  int recursivoMemoization(int i, int j, Piramide p){
355      int x;
356      if (i == p->tamanho-1 )
357      {
358          return p->matriz[i][j];
359      }
360      else
361      {
362          x = (p->matriz[i][j] + maximo(recursivoMemoization(i+1, j, p), recursivoMemoization(i+1, j+1, p)));
363          p->tabela[i][j] = x;
364          return x;
365      }
366  }
```

3.1.2.3 De tráz pra frente - Programação Dinâmica

Na técnica de trás pra frente, utilizamos programação dinâmica para a resolução do problema da pirâmide, o algoritmo utiliza um método iterativo diferentes dos recursos anteriormente usados para percorrer de maneira bottom-up a pirâmide, o processo consiste na sobrescrita de uma posição da linha superior da pirâmide com o maior valor deste subproblema , de maneira sucessiva até o topo, alcançando assim o solução final. A Figura 6 apresenta o algoritmo ultilizado.

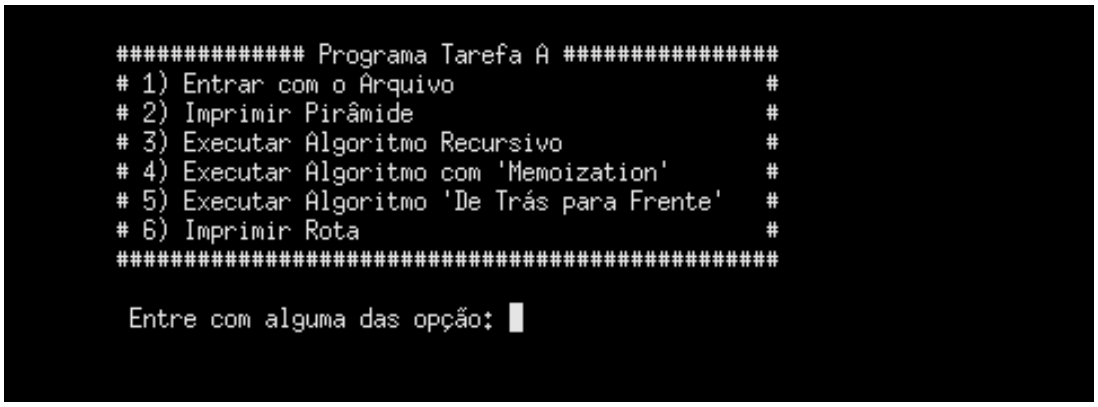
Figura 6: Imprementação da Função ultilizando a técnica deTrazPraFrente, por programação dinâmica.

```
314
315  void deTrazParaFrente(Piramide p){
316      int i,j;
317
318      for(i=p->tamanho - 2; i >= 0; i--)
319      {
320          for(j=0; j <= i; j++)
321          {
322              p->tabela[i][j] = p->tabela[i][j] + maximo(p->tabela[i + 1][j], p->tabela[i + 1][j + 1]);
323          }
324      }
325  }
326
```

3.1.3 Execução e Resultados

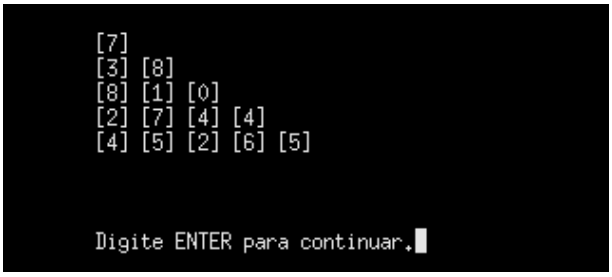
Execução do programa ocorre de maneira bem fácil através da compilação e execução dos arquivos main.c, piramide.c e piramide.h todos sobre a IDE Code Blocks descrita em metodologia, teremos a seguir várias imagens ilustrando a execução em modo Debug, logo na Figura 7 vemos o menu da Tarefa A.

Figura 7: Menu do programa da Tarefa A.



A Partir da opção 1) o usuário pode entrar com o arquivo no formato padrão como já mencionado. Logo após ele tem a opção 2) de imprimir a pirâmide lida como segue na Figura 11, que tem como entrada o exemplo visto no slide em sala de aula.

Figura 8: Resultado da opção 2) Imprimir Pirâmide.



Da opção 3) a 5) o usuário tem a possibilidade de executar os algoritmos proposto na Tarefa A, e obter seus resultados, que além da maior soma do topo até a base mostrar também o tempo gasto. Nas próximas imagens veremos o resultado para cada um dos algoritmos.

Figura 9: Resultado do algoritmo Recursivo Padrão.

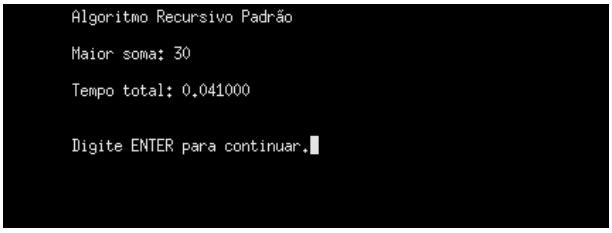
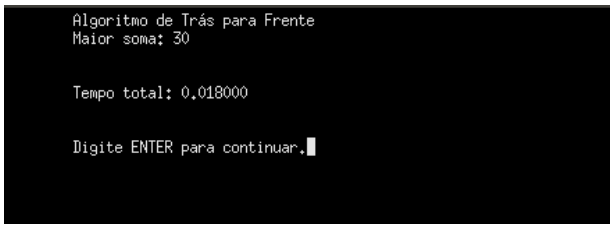


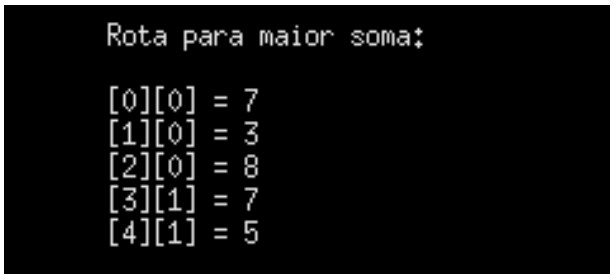
Figura 10: Resultado do algoritmo Memoization.

Figura 11: Resultado do algoritmo, de trás pra frente.



Alem dos resultados para cada algoritmo apresentados, uma rota a partir do resultado da programação dinâmica também é possivel de ser visualizado como segue na Figura 12.

Figura 12: Rota para o melhor resultado.



3.1.3.1 Comparando Resultados

Neste ítem iremos tratar de compara os algoritmos implementos para pirâmides de tamanhos de niveis diferentes sendo 5 niveis para o slide.txt, 10 niveis para o t1.txt e 15 niveis para t2.txt. A Partir desta entradas os resultados apresentados apontam que para valores pequenos de entrada o algoritmo recursivo padrão pode ser eficiente, mas a partir do momento que o tamanho da entradas (níveis da pirâmide) cresce, o resultado apresenta-se bem pior comparado aos outros algoritmos.

Já no algoritmo recursivo com Memoization temos uma tabela para evitarmos retrabalho para algumas soluções, assim deixando o método mais eficiente que o anterior, mas por possuir uma natureza recursiva ainda pode ser bem restritivo.

Agora no último algoritmo analisado, algoritmo de trás para frente, foi possível aplicar as técnicas de programação dinâmica de modo iterativo, o que tornou os resultados

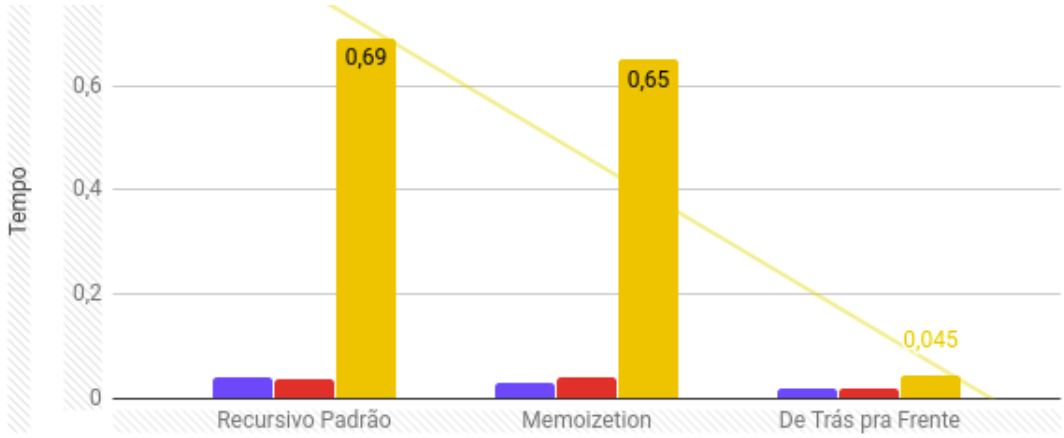
muito mais satisfatórios à medida que os níveis da pirâmide aumentavam. Na próxima Figura 13 podemos ver a média dos resultados obtidos em uma tabela.

Figura 13: Resultados dos algoritmos implementados.

Entrada	Niveis	Recursivo Padrão	Memoizetion	De Trás pra Frente
slide.txt	5	0,041	0,031	0,018
t1.txt	10	0,039	0,041	0,021
t2.txt	15	0,69	0,65	0,045

Agora na Figura 14 podemos ver um gráfico gerado com as entradas da tabela, em azul podemos ver o resultado para uma pirâmide de 5 níveis, vermelho 10 níveis e amarelo 15 niveis. No gráfico fica claro que a partir que o tamanho da entrada aumenta o algoritmo de trás pra frente apresenta um resultado muito melhor, na imagem podemos ver ainda a linha de tendência dos valores.

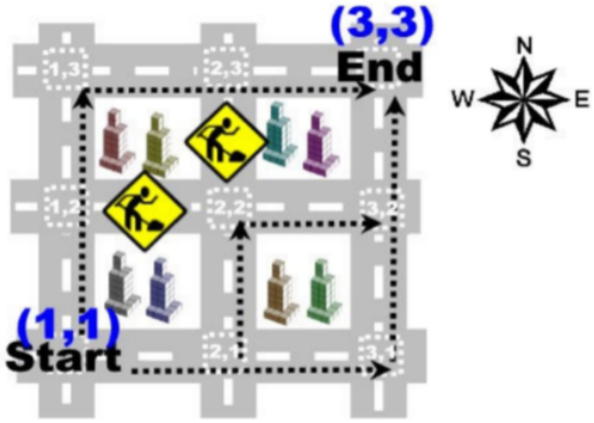
Figura 14: Resultados dos algoritmos implementados.



3.2 Tarefa B

3.2.1 Descrição da Tarefa B

A tarefa se trata de um problema já comentado em sala, no qual e passado uma cidade com diversas rotas e nestas rotas há algumas obras. Assim, pede que achemos os caminhos para se pecorrer entre dois pontos já selecionados na cidade desviando das obras.



3.2.2 Implementação

No tocante a implementação foi usado um TAD, no qual esse possuia uma matriz da cidade com as dadas obras, 2 pontos para representar o tamanho da ciadade, os pontos iniciais da rota e os finais.

```
typedef struct {
    char **ruas;
    int x,y;
    int xi,yi;
    int xf,yf;
}Quadriculado;
```

Além do TAD, para se resolver o problema foi usado outras duas matrizes "count"e "clone"a primeira tinha a função de armazenar as possibilidades de se resolver o problema. A segunda tinha a função de analisar uma das possibilidades qualquer geradas pelo "count"e mostrar ao usuario.

Também e importante notar que para a implentaçao do "clone"foi necessario, criar uma função recussiva "Caminho"que tinha como trabalho seguir um rota da matriz "count"e apontar no "clone".

Figura 15: Imprementação do count

```
count[city->xf][city->yf]=1;
for(i=city->xf;i>=city->xi;i--){
    for(j=city->yf;j>=city->yi;j--){
        if(i<city->x && city->ruas[i][j]!='N')
            count[i][j]=count[i][j]+count[i+1][j];
        if(j<city->y && city->ruas[i][j]!='L')
            count[i][j]=count[i][j]+count[i][j+1];
    }
}
```

Figura 16: Imprementação do clone

```
for(i=1;i<=city->x;i++){
    for(j=1;j<=city->y;j++){
        clone[i][j]='O';
    }
}

Caminho(*city,count,clone,city->xi,city->yi);

for(i=1;i<=city->x;i++){
    for(j=1;j<=city->y;j++){
        printf("%c ",clone[i][j]);
    }
    printf("\n");
}
```

Figura 17: Imprementação do Caminho

```
int Caminho(Quadriculado city,int count[][city.y+1],char clone[][city.y+1],int i,int j){
    if(i>city.x || j>city.y)
        return 0;
    if(i==city.xf && j==city.yf) {
        clone[i][j] = 'X';
        return 1;
    }
    else if(count[i][j]!=0){
        clone[i][j] = '-';
        if(Caminho(city,count,clone,i+1,j))
            return 1;
        else if(Caminho(city,count,clone,i,j+1))
            return 1;
        else
            return 0;
    }
    return 0;
}
```

3.2.3 Execução e Resultados

Ao executar o progama são apresentado 5 opções.No qual exceto a 1 e bloquadas até a leitura do primeiro arquivo.

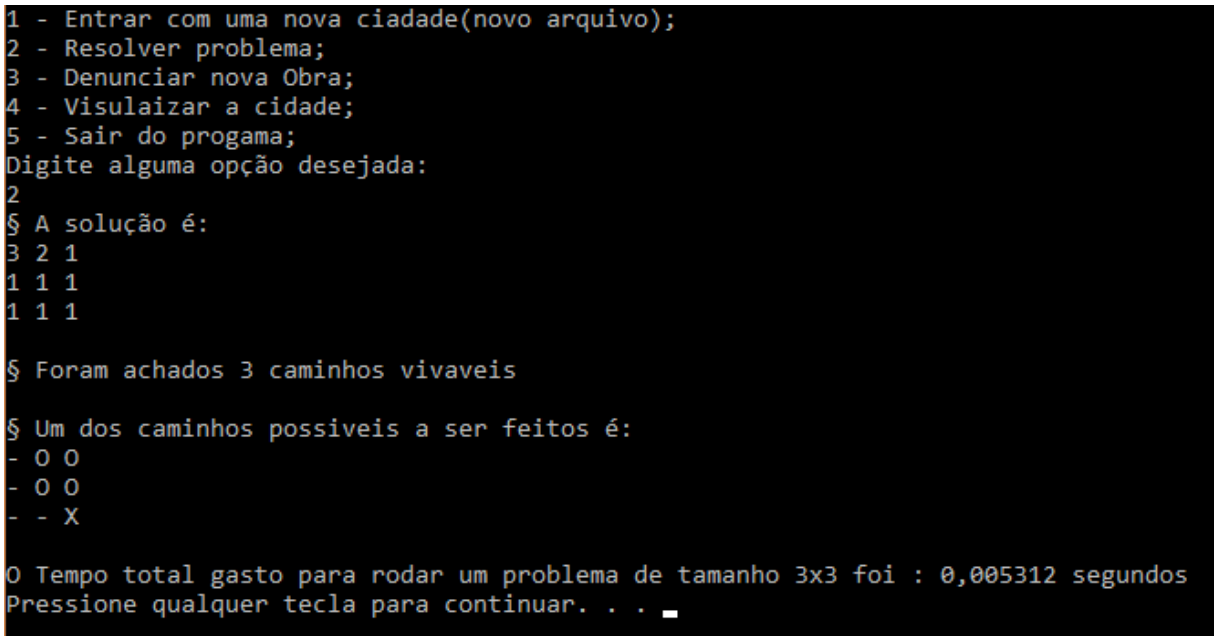
Figura 18: Menu principal

```
---Seja Bem vindo ao problema da cidades---
1 - Entrar com arquivo;
---Bloqueado até a leitura do arquivo---
2 - Resolver problema;
3 - Denunciar nova Obra;
4 - Visulaizar a cidade;
5 - Sair do progama;
----------------------------------------
Digite alguma opção desejada:
```

Após selecionar o primeiro caminho, há a possibilidade de visualizar como está a ciadade e suas obras, além também de adicionar novas obras a cidade ou até mesmo colocar uma nova ciadade.Mas como enfoque há a resoluçao do problema, que ao ser chamado mostra a matriz de possibilidades, a quantidade de caminhos viaveis e uma rota viavel.

Vale ressaltar que o progama também possue um modo debug, podendo ser ativado

Figura 19: Menu principal



em seu codigo fonte por meio de um define no arquivo "main.c". Após ser ativo esse modo permite visualizar o tempo parcial do progama sendo só resolvido a quantidade de caminhos possiveis, já no tempo total engloba-se o tempo parcia mais o tempo gasto para achar uma soluçao valida e mostrar ao usuario.

3.2.4 Comparando resultados

Nesse topico vamos comparar os resultados gerados pelo progama em vista das diversas de diversas entradas. Fora usadas 4 entras uma matriz 3x3,10x10,20x20 e 30x30.

E notavel que a contrução desse problema se deu em complexidade exponecial pelo crescimento da curva, mas isso se da principalmente devido a função "caminho"que procura um caminho valido na ciade para mostrar ao usuario.

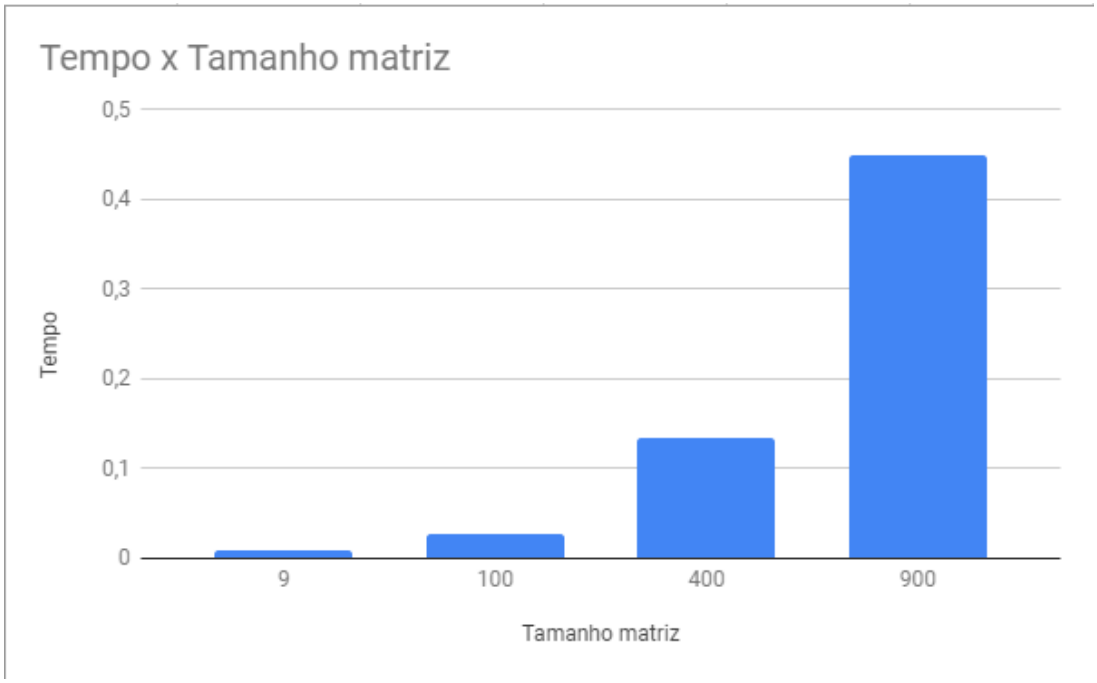
Figura 20: Resuldtados Debug

```
1 - Entrar com uma nova ciadade(novo arquivo);
2 - Resolver problema;
3 - Denunciar nova Obra;
4 - Visulaizar a cidade;
5 - Sair do progama;
Digite alguma opção desejada:
2
§ A solução é:
2045 1253 791 329 119 35 7 1 0 0
792 462 462 210 84 28 7 1 0 0
792 462 252 126 56 21 6 1 0 0
330 210 126 70 35 15 5 1 0 0
120 84 56 35 20 10 4 1 0 0
36 28 21 15 10 6 3 1 0 0
8 7 6 5 4 3 2 1 0 0
1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

§ Foram achados 2045 caminhos vivaveis


O Tempo parcial para rodar um problema de tamanho 10x10, sem um caminho possivel foi : 0,012013 segundos

§ Um dos caminhos possiveis a ser feitos é:
- - - - - - O O O O
O O O O O - - - O O
O O O O O O O - O O
O O O O O O O - O O
O O O O O O O - O O
O O O O O O O - O O
O O O O O O O - O O
O O O O O O O X O O
O O O O O O O O O O
O O O O O O O O O O

O Tempo total gasto para rodar um problema de tamanho 10x10 foi : 0,025119 segundos
Pressione qualquer tecla para continuar. . .
```

Figura 21: Tabela de tamanhos por tempo

Tamanho matriz	Tempo
9	0,007983
100	0,026704
400	0,134048
900	0,448771
900v2	x

Figura 22: Grafico de tamanhos por tempo



Conclusão

Através do trabalho realizado foi possível perceber que é de essencial importância a aplicação correta da técnica a ser utilizada, pois dependendo do contexto algum dos algoritmos estudados pode-se torna inviável, logo uma boa modelagem é algo fundamental. Além disso percebemos que a técnica de programação dinâmica pode nos auxiliar muito na resolução de problemas para grandes entradas de dados, algo importante já que na maioria dos casos estamos interessados em entradas que possuem esta característica.

Referências

ZIVIANI, N. *Projeto de Algoritmos*. 3 edição. ed. [S.l.]: Cengage Learning, 2010. Citado na página 6.