

Trabalho Prático 3: Plano de Dominação Global do professor WM. Jr

Algoritmos e Estruturas de Dados III – 2016/2
Entrega: 23/01/2017

1 Introdução

O professor WM. está tramando um plano para dominar todas as cidades do planeta, tornando assim, o líder supremo do planeta. Para ajudar o plano de dominação global, WM. conta com seus pupilos, os mestres do mal *CDFDR*. O plano é bem simples, os mestres do mal conseguem atacar uma cidade coma arma de manipulação mental e convertem todos da cidade a adorar o professor WM. porém, essa arma tem efeitos colaterais “áridos”, toda vez que ela é usada todas as cidades próximas explodem. O professor quer o maior número de seguidores possíveis, então ele quer dominar as cidades de tal forma a ter a maior população possível.¹

Você foi escolhido para ajudar o professor WM. a decidir quais cidades ele irá atacar, dado o grid $N \times M$ de todas as cidades do planeta com suas respectivas populações o objetivo é recuperar a soma da maior população possível. a destruição da sua arma não é tão intuitiva, dado o grid $N \times M$, se a cidade $C_{i,j}$ é dominada as cidades $\{C_{i,j+1}; C_{i,j-1}; C_{i+1,k}; C_{i-1,k}\} \forall k \in \{1, \dots, M\}$ são destruídas, em outras palavras, as cidades das colunas ao lado e todas as cidades da linha acima e abaixo a escolhida são destruídas. O WM. é brilhante e já sabe como resolver esse problema, mas não te contou por motivos educacionais, mas deu a dica que a solução é por programação dinâmica (ele não vai aceitar solução sem ser por programação dinâmica).

O professor WM. tem uma ideia, mas ela te dará mais trabalho. Como a cidade tem muitas cidades (dizem que tem tantas cidades quanto número de estrelas no espaço observáveis $9 \cdot 10^{21}$), mesmo que implementando o programa de maneira eficiente ele ainda será lento. O professor tem um laboratório com máquinas bem potentes (vários núcleos de processamentos), então ele quer que o código seja paralelo, ele gosta bastante da linguagem *C* então

¹Trabalho baseado no problema C da final sul americana de 2008

o paralelismo deve ser usando *pthread*s, para facilitar o trabalho coloque o número de threads como um parâmetro de entrada do programa (*argv[1]*).

O trabalho então consiste das seguintes tarefas:

- Dada uma matriz $N \times M$ contendo a população de cada cidade, descobrir qual a população máxima a recuperar usando **estratégia de programação dinâmica** seguindo as restrições de escolha de cidades;
- Paralelizar a solução de programação dinâmica usando *pthread*s.

2 Entrada e Saída

A entrada e saída é a padrão do C, *stdin* e *stdout*, não é necessário abertura de arquivos para leitura da entrada ou escrita da saída, o número de threads deve entrar como um parâmetro do programa, isto é, a função *main* do programa deve receber dois parametros da seguinte forma:

```
1  int main(int argc, char *argv[]) {
2      ... \\aqui vem seu codigo
3      return 0;
4  }
```

O programa deverá conter um unico argumento, um número inteiro que indique o número de threads que o programa usará, garantimos que esse número nunca será maior que o número de threads máximo do computador de teste.

Entrada A entrada consiste em 2 inteiros positivos M e N ($1 \leq M \times N \leq 10^4$), separados por espaço. M e N representam o número de linhas e colunas, respectivamente. A entrada segue com mais M linhas, cada uma contendo N inteiros separados por espaços. Cada um desses inteiros representam a população de cada cidade. Pode-se garantir que cada cidade tem ao menos população maior ou igual a 1 e menor ou igual a 100.

Saída A saída deve ter apenas uma linha contendo um único número inteiro que representa a população máxima que é possível dominar do grid.

Exemplo de execução Seja o seguinte grid de entrada

1	8	2	1	9
1	7	3	5	2
1	2	10	3	10
8	4	7	9	1
7	1	3	1	6

Uma sequencia ótima de escolhas para esse problema seria

1	8	2	1	9	1	8	2	1	9	1	8	2	0	0	0	0	0	0	0
1	7	3	5	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	10	3	10	1	0	0	0	10	1	0	0	0	10	1	0	0	0	10
8	4	7	9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	3	1	6	7	1	3	1	6	7	1	3	1	6	7	1	3	1	6

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	10	1	0	0	0	10	1	0	0	0	10	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	6	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0

sendo assim a saída nesse caso seria uma única linha com o valor **54**.

3 O que deve ser entregue

Deverá ser submetido um arquivo **.zip** contendo somente uma pasta chamada **tp2** e dentro desta deverá ter: (i) Documentação **em formato PDF** e (ii) Implementação.

Documentação Poderá ter no máximo 10 páginas e deverá seguir tanto os critérios de avaliação discutidos na Seção 4.1, bem como as diretrizes sobre a elaboração de documentações disponibilizadas no *moodle*. Além disso, a documentação deverá conter análise experimental validando as complexidades de tempo e espaço.

Implementação Código fonte do seu TP (*.c* e *.h*).

Makefile Inclua um *makefile* na submissão que permita compilar o trabalho. É obrigatório o uso das *flags*: **-Wall -Wextra -Werror -std=c99 -pedantic** na compilação.

4 Avaliação

Eis uma lista **não exaustiva** dos critérios de avaliação que serão utilizados.

4.1 Documentação

Introdução Inclua uma breve explicação do problema que está sendo resolvido no seu trabalho e um resumo da sua solução.

Solução do Problema Você deve descrever a solução do problema de maneira clara e precisa, detalhando e justificando os algoritmos e estruturas de dados utilizados. Para tal, artifícios como pseudo-códigos, exemplos ou diagramas podem ser úteis. Note que documentar uma solução não é o mesmo que documentar seu código. **Não** é necessário incluir trechos de código em sua documentação nem mostrar detalhes de sua implementação, exceto quando estes influenciem o seu algoritmo principal, o que se torna interessante.

Análise de Complexidade Inclua uma análise de complexidade de tempo e espaço dos principais algoritmos e estrutura de dados utilizados. Cada complexidade apresentada deverá ser devidamente **justificada** para que seja aceita.

Avaliação Experimental Sua documentação deve incluir os resultados de experimentos que avaliem o tempo de execução de seu código em função de características da entrada. Cabe a você gerar entradas para esses experimentos. Por exemplo: se esse trabalho fosse sobre ordenação, seria interessante mostrar como o tempo de execução de cada algoritmo varia quando o número de itens a serem ordenados aumenta. Para tal, um gráfico mostrando o tempo de execução em função do tamanho da entrada pode ser interessante. Você também deve interpretar os resultados obtidos. Comente sobre cada gráfico ou tabela que você apresentar mostrando o que é possível concluir a partir dele.

4.2 Implementação

Linguagem & Ambiente O seu programa deverá ser implementado na linguagem **C** e poderá fazer uso de funções da biblioteca padrão da linguagem. Trabalhos que utilizem qualquer outra linguagem de programação e/ou que façam uso de outras bibliotecas que não a padrão serão zerados. Além disso, certifique-se que seu código compile e funcione corretamente nas máquinas **Linux** dos laboratórios do DCC.

Casos de teste A sua implementação passará por um processo de correção automatizado, portanto, o formato da saída do seu programa deve ser idêntico aquele descrito nessa especificação. Saídas com qualquer divergência serão consideradas erradas, mesmo que as divergências sejam *whitespaces*. e.g. espaços, *tabs*, quebras de linha, etc. Para auxiliá-lo na depuração do seu código, será fornecido um pequeno, **não-exaustivo**, conjunto de entradas e suas respectivas saídas. É seu dever certificar-se que seu código funciona corretamente para qualquer entrada válida.

Alocação Dinâmica Algoritmos e estruturas de dados deverão fazer uso de memória alocada dinamicamente (`malloc()` ou `calloc()`). Certifique-se que seu programa utiliza essas regiões de memória corretamente, pois os monitores penalizarão implementações que realizam *out-of-bounds access* e que tenham vazamento de memória (não desalocar memória dinâmica). A alocação dinâmica deverá fazer uso das funções `malloc()` ou `calloc()` da biblioteca padrão C, bem como liberar tudo o que for alocado utilizando `free()`, para gerenciar o uso da memória. **DICA:** Utilize `valgrind` antes de submeter o seu TP.

Qualidade do código Seu código também será avaliado no quesito de legibilidade, dando atenção, porém não limitando-se, aos seguintes itens: (i) **INDENTAÇÃO**; (ii) nomes de variável e função descritivos e claros; (iii) Modularização adequada; (iv) Comentários dentro de funções, explicando o que certos trechos mais complicados fazem; (v) Comentários fora de funções, explicando, em alto-nível, o que as funções mais importantes fazem; (vi) funções concisas que desempenham somente uma tarefa; (vii) **Proibido uso de variáveis globais**.

Atrasos Trabalhos poderão ser entregues após o prazo estabelecido, porém sujeitos a uma penalização regida pela seguinte fórmula:

$$\Delta_p = \frac{2^{d-1}}{0.32} \%$$

Por exemplo, se a nota dada pelo corretor for 70 e você entregou o TP com 4 dias corridos de atraso, sua penalização será de $\Delta_p = 25\%$ e, portanto, a sua nota final será: $N_f = 70 \cdot (1 - \Delta_p) = 52.2$. Note que a penalização é exponencial e 6 dias de atraso resultam em uma penalização de 100%.

5 Consideração Final

Assim como em todos os trabalhos dessa disciplina é estritamente proibida a copia parcial ou integra de códigos, seja da internet ou de colegas. Utilizaremos o algoritmo *MOSS* para detecção de plágio em trabalhos, seja honesto. Você não aprende nada copiando código de terceiros nem pedindo a outra pessoa que faça o trabalho por você. Se a cópia for detectada, sua nota será zerada e os professores serão informados para que as devidas providências sejam tomadas.

HAVE FUN!!!