Trabalho Prático 1: Where are the Panzers?

Entrega: 06/05/2015

1 Introdução

O ano é 1944 e o Brasil está na Segunda Guerra Mundial. Você está em uma equipe dos Aliados encarregada de decodificar mensagens do Eixo, sob o comando do General Morais. O único dispositivo que sua equipe possui é uma antena que consegue captar ondas de rádio enviadas de um centro de Eixo para sua respectiva força aérea. Sua equipe recebe vários sinais na antena mas não consegue entender nada do que se passa.

Para a sorte de sua equipe, o espião Fonseca conseguiu reunir algumas informações importantes. A primeira informação é sobre como decodificar as ondas de rádios recebidas. Fonseca revelou que o Eixo utiliza de um sistema representado por 0 e 1's e estes bits estão codificados nas ondas de rádio. Ele mostrou como recuperar esses valores das ondas de rádio e como interpretá-los. Para a surpresa de sua equipe, o que estava sendo transmitido pelo canal eram caracteres alfanuméricos.

Como uma amostra, sua equipe decodificou algumas sequências de ondas de rádio. Um dos exemplos amostrados foi:

jahe?! $paeaoaew11iawwnaaqtaaa(aqw1o0o0,jajaja9aq0qqq1aaa)xx

O espião Fonseca, como um dos melhores espiões dos aliados, conseguiu as instruções para extrair a informação dessa sequência de caracteres. Ele mostrou que a mensagem continha a coordenada (100,901) como alvo de ataque, conforme mostrado abaixo:

jahe?! $**p**aea**o**aew11**i**aww**n**aaq**t**aaa**(**aqw**1**o**0**o**0,**jajaja**9**aq**0**qqq**1**aaa**)**xx

Um documento roubado pelo espião mostra as regras para extrair as coordenadas. As regras são descritas abaixo:

- Uma coordenada sempre virá precedida da expressão "p[o]int", minúscula. Essa expressão indica que os caracteres que estão fora do colchete aparecem apenas uma vez, na ordem mostrada. O caractere "o", entretanto, pode aparecer uma ou mais vezes. Dessa forma, as strings "point" e "pooooint" são válidas mas "pint" não é.

- A coordenada do ataque é no formato de coordenadas cartesianas “(x,y)”, onde x e y são números inteiros no intervalo de 0 (inclusive) a 10^5 (exclusive). Além disso, você deve ignorar 0’s à esquerda. Por exemplo, 0050 representa o número 50.
- O número máximo de ’o’, em sequência, é 2^{31}.
- O processo para identificar as coordenadas ocorre em duas etapas: descobrir a expressão “p[o]int” e descobrir a coordenada “x,y”.
- Na primeira etapa, a expressão “p[o]int” pode estar com ruídos entre seus caracteres. O ruído pode ser qualquer caracter da tabela ascii e que seja imprimível[1].
- Não existe um número definido de ruídos entre as coordenadas. Neste caso, esses ruídos devem ser ignorados a fim de recuperar as coordenadas.
- Caracteres da expressão podem virar ruídos após serem lidos, como o exemplo “pooooiooont(20,30)”. Após ler o caractere “p”, são lidos 4 caracteres “o” e um caractere “i”. Os caracteres “o” são ruídos pois a meta é encontrar o caractere “n”. Portanto eles são desconsiderados. Uma implicação disso é que eles não são considerados para a contagem de tanques.
- Uma vez que a palavra “point” seja descoberta, inicia-se a segunda etapa de descobrir a coordenada. Enquanto um parenteses de abertura (“(”) não for encontrado, a segunda etapa não inicia-se. A segunda etapa é invalidada e o processo voltará para o início da primeira etapa se:
  - for lido mais de um parenteses de abertura (“(”).
  - for lido mais de uma vírgula (“,”).
  - o valor máximo da coordenada for ultrapassado.
- Assim que ocorrer uma violação em uma das regras acima, o processo deve ser resetado, ou seja, deve-se começar a procura da letra “p” novamente.
- Podem haver ruídos entre os algarismos dos números inteiros x e y bem como entre o padrão “(x,y)”. Esses ruídos deverão ser desconsiderados.
- Uma mesma coordenada pode aparecer mais de uma vez, o que indica que houve mais de um ataque a esta mesma coordenada. Além disso, o número de tanques em cada uma dessas aparições pode variar.

A primeira função sua, como especialista em algoritmos, é criar uma solução que extraia as coordenadas de uma sequência de caracteres, bem como o número de tanques Panzer que serão designados para aquela coordenada. Para identificar o número de tanques Panzer que serão utilizadas no ataque, o espião Fonseca descobriu que é o número de “o” da expressão “p[o]int” que

[1] http://pt.wikipedia.org/wiki/ASCII

identifica a coordenada. Dessa forma, "pooooint(50,40)" indica que a coordenada (50,40) será alvo de 4 tanques Panzer enquanto que "point(30,11)" indica que a coordenada (30,11) será alvo de apenas um tanque.

O general Morais deseja enviar reforços para as coordenadas que serão alvos de ataque. Como o número de tanques disponíveis são finitos, o general definiu alguns critérios para determinar quais coordenadas ele irá ajudar. O primeiro critério é ajudar as coordenadas que serão atingidas por mais tanques pois essas são mais críticas. Como segundo critério, se duas coordenadas forem alvo do mesmo número de tanques, a que estiver mais próximo da base onde você está localizado será atendida primeiro. Se a distância também for igual, a primeira coordenada detectada na interceptação será auxiliada primeiro.

Por exemplo, suponha que as coordenadas (10,30) e (10,10) apareceram nesta ordem e irão ser atacadas pelo mesmo número de tanques. Se a coordenada da sua base for (10,20), as coordenadas também empatarão na distância e, como especificado no último critério, a coordenada (10,30) virá antes no relatório pois apareceu antes da coordenada (10,10) na entrada. Agora suponha que a coordenada (10,30) irá ser atacada por um número menor de tanques que a coordenada (10,10). Neste caso, a coordenada (10,30) virá após a coordenada (10,10) no relatório pois o número de tanques que irá atacá-la é menor.

Com esses critérios, o general ordenou que você produza um relatório das coordenadas, em ordem decrescente de prioridade. Para simplificar a situação, dado que o matemático da equipe morreu no último bombardeio, a proximidade poderá ser medida com distância euclidiana. Você, como um hábil programador, sabe que os computadores da base possuem uma **quantidade pequena de memória primária e que, muito provavelmente, as informações das coordenadas não serão comportadas nesta memória de uma vez**. Por isso, você sabe que seu programa deverá ser pensado para contornar essa situação e utilizar o disco. Se não houver nenhuma coordenada, o relátorio deverá ser em branco, ou seja, nenhum caractere deverá ser impresso.

2 Código

Você deve implementar, em linguagem C, um programa que seja capaz de:

- Dado um conjunto de caracteres ASCII imprimíveis[2] e o caractere de quebra de linha (\n), ser capaz de extrair as coordenadas que serão alvos de tanques Panzer, bem como o número de tanques Panzer que serão enviados para a coordenada.
- Emitir um relatório em ordem decrescente de prioridade.
- Operar com uma quantidade limitada de memória X.

Enfatizando o último item, seu programa receberá uma quantidade de memória limitada X para sua executação. Essa quantidade será para comportar **TODOS** os recursos do seu programa, que **inclui o binário (código e alocações estáticas) e o heap, para alocações dinâmicas**. Dessa

[2] http://pt.wikipedia.org/wiki/ASCII

forma, não somente a ordenação deverá executar sobre tais condições mas a leitura da entrada também. **Tenha cuidado com o tipo de função que você usará para ler o arquivo de entrada.**

Nós limitaremos o uso de memória do processo onde executará o seu código. Dessa forma, se seu código ultrapassar esse limite, ele não será mais capaz de utilizar a memória, podendo levar a erros. Nós informaremos esse limite pela entrada padrão. **Tenha ciência que essa informação é apenas para ajudar você a administrar os recursos de memória**. O ambiente em que os testes serão executados já estarão configurados com essa limitação.

Uma pergunta pertinente é como você pode fazer para testar o seu programa. Em Linux, para limitar o tamanho da memória de programa, você pode utilizar a chamada de sistema "setrlimit"[3]. A sintaxe do "setrlimit" recebe dois parâmetros: o recurso que você quer limitar (resource) e uma estrutura com as limitações (struct rlimit). O recurso que será limitado é o AS, que é a memória virtual, em bytes, do processo. As limitações possuem dois valores: "soft" e "hard". A primeira é o quanto o kernel do sistema operacional irá limitar. Se esse limite for ultrapassado, o kernel verificará o segundo valor. Se a quantidade de memória que ultrapassou o limite "soft" ainda for menor que o limite "hard", o kernel continuará executando o processo. Se for maior, você não conseguirá mais alocar nada e seu programa irá parar de funcionar corretamente. Como nos testes não haverá tolerância, tanto "soft" quanto "hard" serão iguais.

Nós recomendamos que você utilize a função abaixo para restringir o tamanho da memória do seu processo:

```
#include <sys/time.h>
#include <sys/resource.h>
#include <stdint.h>

void SetMemoryLimit(const int64_t limit_mb)
{
   struct rlimit limits;
   limits.rlim_cur = limit_mb * 1024ull * 1024ull;
   limits.rlim_max = limit_mb * 1024ull * 1024ull;
   int error_code = setrlimit(RLIMIT_AS, &limits);
}
...

int main(){
   SetMemoryLimit(10ull);
   ... Seu programa ...
   return 0;
}
```

O argumento recebido pela função é a quantidade de memória, em MB, que você deseja limitar. No exemplo acima, o programa em questão está com um limite de 10 MB para executar. Uma nota importante é que seu programa pode não terminar quando você ultrapassar o limite mas pode funcionar incorretamente. Por exemplo, se você tentar alocar 20MB com o "malloc", este retornará

[3]Uma fonte mais detalhada se encontra em http://linux.die.net/man/2/setrlimit

um ponteiro para "NULL" e tudo que depender dessa alocação não irá funcionar (provavelmente irá dar um erro de falha de segmentação).

Observe que essa função está sendo passada para que você teste seu programa. Nos testes que serão executados pelos monitores, o processo o qual executará seu programa já será limitado. Logo, não há necessidade de você limitar a memória. A frase que se segue é para aqueles que tentarem burlar o sistema já pouparem tempo: a limitação imposta no processo onde seu programa executará sobrepõe a limitação interna do seu programa. Logo, se você tentar configurar um limite maior que o nosso, não irá conseguir.

Por fim, como o sistema PRATICO não oferece o recurso de limitar a memória, ele será utilizado apenas para verificar se sua saída está correta para os testes preliminares. O restante dos testes bem como os testes do PRATICO serão executados, com limitação de memória, durante as entrevistas.**Dessa forma, os acertos no PRATICO só serão considerados caso eles executem durante a entrevista. Obviamente, se você tentar solucionar o problema não considerando a limitação de memória, você poderá não acertá-los na entrevista e ter um percentual de acerto extremamente baixo nos testes. A dica é: não vá para este caminho.**

3 Entrada e Saída

Seu programa deverá ler a entrada da entrada padrão (`stdin`) e gravar a saída na saída padrão (`stdout`). A entrada de teste é composta apenas por um caso de teste. A primeira linha contém um inteiro que corresponde ao tamanho máximo que seu programa receberá para executar, em Megabytes (1 Megabyte = 1024^2 bytes). O tamanho mínimo de memória que seu programa receberá será de 30MB, de forma a comportar, pelo menos, o binário e as alocações estáticas. Faça uso dessa informação pois, dependendo do número de coordenadas que seu programa encontrar, você precisará utilizar bem essa memória alocada para você, de forma que seu programa não ultrapasse o tempo limite de execução de cada teste. **Lembre-se que você poderá utilizar o disco como memória externa, que contém um espaço bem maior de armazenamento**.

A segunda linha contém 2 inteiros, separados por espaço, que correspondem, respectivamente, aos valores x e y da coordenada de sua base. As próximas linhas são compostas **um número não definido de caracteres imprimíveis da tabela ASCII**, ou seja, o tamanho da entrada pode ser bem grande, ultrapassando facilmente a barreira de GB. A saída deverá ser composta de um par T;C, onde T é o número de tanques Panzer que atacará a coordenada C, onde C é uma coordenada com formato (x,y).

4 O que entregar

Você deve submeter uma documentação de até 10 páginas contendo uma descrição de sua solução para o problema, além de uma análise de complexidade de tempo dos algoritmos envolvidos e uma análise da memória gasta pelo seu programa. Siga as diretrizes sobre como fazer uma documentação que foram disponibilizadas no portal *minha.ufmg*.

ENTRADA	SAÍDA
30 15 15 uehuaheuahuhaupkkkkokkki kkknnkkktk(8i0,9p0)uahau heauheuaheuhueauehuaheua huhaukkk21321kkkikkknkkk tk(8i0000!!!!0000000,9a0)uahauheauheuaheuhueapoi nt(10,20)pooooiont(20,30)poooaaaooooooooint(30,4 0)poo??oint(40,50)uakkka kakijeuquueuhupoint(poin t100,200)[[[[[]]]]]point (,30)point(100,100)%421 11point(11point(11,11))1 1:19point(000000010,1)	11;(30,40) 4;(20,30) 3;(40,50) 1;(10,20) 1;(10,1) 1;(80,90) 1;(100,100) 1;(100,200)

Table 1: Toy example: teste o qual o TP deve gerar a saída correta para ser elencado para uma entrevista.

ENTRADA	SAÍDA
30 40 10 ppppooooooooooooooooi(11, 14)nt(40,11)pooint(100000 ,10)point(1000000,50)pooo ooooooooooooointpointpoin tpointpoint(40,9)	16;(40,11) 16;(40,9)
35 1000 1000 poooooooooooooooooooooooo ooooooooooooooooooooooooo ooooooooooooooooooooooooo ooooooooooooooooooooooooo ooooooooooooooooooooooooo int(1000,1001)point(1,1,) point((1,1),pint(10,10)?? aaaaa aa”””””””apointk(,10) pooooint(1000,1002)lllaa	124;(1000,1001) 5;(1000,1002)

Table 2: Outros exemplos de teste.

Além da documentação, você deve submeter um arquivo compactado no formato *.zip* contendo todos os arquivos de código (*.c* e *.h*) que foram implementados. Além dos arquivos de código, esse arquivo compactado deve incluir um *makefile*. Consulte os tutoriais disponibilizados no *minha.ufmg* para descobrir como fazer um. Finalmente, lembre-se de não incluir nenhuma pasta no arquivo compactado.

5 Avaliação

Seu trabalho será avaliado quanto a documentação escrita e à implementação. Eis uma lista **não exaustiva** de critérios de avaliação utilizados.

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. Para tal, artificios como pseudo-códigos, exemplos ou figuras podem ser úteis. Note que documentar uma solução não é o mesmo que documentar seu código. **Não** é preciso incluir trechos de código em sua documentação nem mostrar detalhes de sua implementação, exceto quando os mesmos influenciem o seu algoritmo principal, o que se torna interessante.

Análise de Complexidade Inclua uma análise de complexidade de tempo dos principais algoritmos implementados e uma análise de complexidade de espaço das principais estruturas de dados de seu programa. 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 items 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.

Limite de Tamanho Sua documentação deve ter no máximo 10 páginas. Todo o texto a partir da página 11, se existir, será desconsiderado.

Guia Há um guia e exemplos de documentação disponíveis no moodle.

Implementação

Linguagem e Ambiente

- Implemente tudo na linguagem **C**. Você pode utilizar qualquer função da biblioteca padrão da linguagem em sua implementação, mas **não** deve utilizar outras bibliotecas. **Trabalhos em outras linguagens de programação serão zerados. Trabalhos que utilizem outras bibliotecas também.**
- Os testes serão executados em **Linux**. Portanto, garanta que seu código compila e roda corretamente nesse sistema operacional. A melhor forma de garantir que seu trabalho rode em Linux é escrever e testar o código nele. Há dezenas de máquinas com Linux nos laboratórios do DCC que podem ser utilizadas. Você também pode fazer o download de uma variante de Linux como o **Ubuntu** (`http://www.ubuntu.com`) e instalá-lo em seu computador ou diretamente ou por meio de uma máquina virtual como o **VirtualBox** (`https://www.virtualbox.org`). Há vários tutoriais sobre como instalar Linux disponíveis na web.

Casos de teste Seu trabalho será executado em um conjunto de entradas de teste.

- Essas entradas *não* serão disponibilizadas para os alunos até que a correção tenha terminado. Faz parte do processo de implementação testar seu próprio código.
- Você perderá pontos se o seu trabalho produzir saídas incorretas para algumas das entradas ou não terminar de executar dentro de um tempo limite pré-estabelecido. Esse tempo limite é escolhido com alguma folga. Garanta que seu código roda a entrada de pior caso em no máximo alguns minutos e você não terá problemas.
- A correção será **automatizada**. Esteja atento ao formato de saída descrito nessa especificação e o siga precisamente. Por exemplo: se a saída esperada para uma certa entrada o número 10 seguido de uma quebra de linha, você deve imprimir apenas isso. Imprimir algo como "A resposta e: 10" contará como uma saída **errada**.
- Os exemplos mostrados nessa especificação são parte dos casos de teste.
- Os testes disponíveis no PRATICO são apenas preliminares. Os monitores executarão os trabalhos nos demais casos de teste.
- **Você deve entregar algum código e esse código deve compilar e executar corretamente para, pelo menos, o *toy example* da tabela 1, que é o primeiro teste do PRATICO. Se isso não ocorrer, a nota do trabalho prático será zerada.**

Alocação Dinâmica Você 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 que está disponível para você.

Makefile Inclua um makefile na submissão que permita compilar o trabalho.

Qualidade do Código Seu código deve ser bem escrito:

- Dê nomes a variáveis, funções e estruturas que façam sentido.
- Divida a implementação em módulos que tenham um significado bem definido.
- Acrescente comentários sempre que julgar apropriado. Não é necessário parafrasear o código, mas é interessante acrescentar descrições de alto nível que ajudem outras pessoas a entender como sua implementação funciona.

- Evite utilizar variáveis globais.
- Mantenha as funções concisas. Seres humanos não são muito bons em manter uma grande quantidade de informações na memória ao mesmo tempo. Funções muito grandes, portanto, são mais difíceis de entender.
- Lembre-se de indentar o código. Escolha uma forma de indentar (tabs ou espaços) e mantenha-se fiel a ela. Misturar duas formas de indentação pode fazer com que o código fique ilegível quando você abri-lo em um editor de texto diferente do que foi utilizado originalmente.
- Evite linhas de código muito longas. Nem todo mundo tem um monitor tão grande quanto o seu. Uma convenção comum adotada em vários projetos é não passar de 80 caracteres de largura.
- **Tenha bom senso**.

6 Considerações Finais

- Essa especificação não é isenta de erros e ambiguidades. Portanto, se tiver problemas para entender o que está escrito aqui, pergunte a nós, monitores.
- A penalização por atraso seguirá será de $\frac{2^{d-1}}{0.32}\%$, em que d é o número de dias **úteis** de atraso.
- Você **não** precisa de utilizar uma IDE como Netbeans, Eclipse, Code::Blocks ou QtCreator para implementar esse trabalho prático. No entanto, se o fizer, **não** inclua os arquivos que foram gerados por essa IDE em sua submissão.
- Esteja atento ao tamanho dos arquivos de entrada e ao uso de memória de seus algoritmos. Faça uso da chamada de sistema "setrlimit" para testar seu programa e verificar se ele respeita as limitações de memória sob diversos casos.
- **Os testes para os quais seu programa será submetido terão um tempo limite de execução. Para entradas muito grandes, você receberá mais memória para executar. Utilize essa quantidade extra de memória para fazer seu programa executar mais rapidamente.**
- **A informação contida no arquivo de entrada, com ruídos, poderá chegar até 5GB.**
- **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 tomem as devidas providências.