O projeto valerá de 0 a 10 e será levada em consideração a organização dos códigos de cada solução. Escreva códigos legíveis e com comentários sobre cada decisão importante feita nos algoritmos. As questões deverão ser implementadas em pthreads e utilizando o sistema operacional Linux. Ademais, caso uma questão necessite de arquivos, a equipe deverá disponibilizar arquivos exemplos de entrada. O não cumprimento das regras acarretará em perdas de pontos na nota final.
Atenção: Usar somente pthreads e Linux.

1. Uma cidade começou a usar um novo sistema eletrônico de votação para as eleições municipais. Após o período de votação, os votos de cada bairro estão em um arquivo e precisam agora ser contabilizados. Faça um programa que receba um número ***N*** de arquivos , um número ***T <= N*** de *threads* utilizadas para fazer a contagem, e um número ***C*** de candidatos a prefeito. Em seguida, o programa deverá abrir os ***N*** arquivos nomeados ***"x.in"*** no qual ***1 <= x <= N***. Cada arquivo terá 1 voto por linha que será um número ***y | 0 <= y <= C*** em que 0 significa voto em branco, 1 significa voto ao candidato 1 e assim sucessivamente. Cada *thread* deverá pegar um arquivo. Quando uma *thread* concluir a leitura de um arquivo, e houver um arquivo ainda não lido, a *thread* deverá ler algum arquivo pendente. Ao final imprima na tela o total de votos, a porcentagem de votos recebidos por cada candidato (e dos em branco também) e o número do candidato vencedor (que será o candidato com mais votos não importando a porcentagem).

Assumindo o conhecimento prévido da quantidade de threads e arquivos, pode-se definir no início do programa quais arquivos a serem tratados por cada thread. Uma outra alternativa ler os arquivos sob demanda, a partir do momento que uma thread termina a leitura de um arquivo, pega qualquer outro não lido dinamicamente.

*Ademais, deve-se garantir a exclusão múltua ao alterar o array que guardará os votos de cada candidato. Uma implementação mais refinada garante a exclusão mútua separada para cada posição do array. Mais especificamente, enquanto um voto está sendo contabilizado para um candidato **x** e modificando o array na respectiva posição, uma outra thread pode modificar o array em uma posição **y** que representação outro candidato. Ou seja, se o array de votos possui tamanho 10, haverá um outro array de 10 mutex, um para cada posição do vetor de votos. Ao ler um arquivo e detectar o voto para o candidato y, a thread trava o mutex relativo à posição y, incrementa a quantidade de votos, e destrava o mutex na posição y. Obviamente, se mais de uma thread quiser modificar a mesma posição do array de votos simultaneamente, somente 1 terá acesso, e as outras estarão bloqueadas. O mutex garantirá a exclusão mútua na posição.*

2. Você deverá implementar um sistema computacional de controle de ferrovias usando C e *pThreads*. A ferrovia é composta por 5 interseções, as quais só permitem **2 trens simultaneamente**. Todavia, um trem passando em uma interseção não deve afetar (bloquear) o andamento dos outros trens em outras interseções. Uma interseção é representada por uma variável inteira. Um trem passando pela interseção (com menos de 2 trens) deverá modificar a

variável contador da interseção indicando a quantidade de trens nesta, e esperar 500 milissegundos para indicar o término de sua passagem. Ao concluir a passagem na interseção, deverá decrementar o respectivo contador em uma unidade. O trem, que liberar uma interseção, somente deverá notificar algum trem aguardando a liberação desta interseção.

Por exemplo: Trem 1 e Trem 2 estão na interseção 5, e Trem 3 está aguardando a liberação desta interseção (depois de ter passado pela interseção 4). Trem 1, ao sair da interseção 5, disponibilizará um trilho na interseção, mas só deverá notificar sua saída para o Trem 3 (pois é o único trem aguardando a interseção 5). Os trens deverão ser implementados usando threads, as quais precisam acessar as interseções na sequência 1,2,3,4,5. Ao concluir o percurso, cada trem começará a trafegar novamente a partir do início (ou seja, a partir da interseção 1)**.** Assuma a existência de 10 trens.

1 -2 -3 - 4 - 5

==========>

3. Um sistema gerenciamento de banco de dados (SGBD) comumente precisa lidar com várias operações de leituras e escritas concorrentes. Neste contexto, podemos classificar as threads como leitoras e escritoras. Assuma que enquanto o banco de dados está sendo atualizado devido a uma operação de escrita (uma escritora), as threads leitoras precisam ser proibidas em realizar leitura no banco de dados. Isso é necessário para evitar que uma leitora interrompa uma modificação em progresso ou leia um dado inconsistente ou inválido.

Você deverá implementar um programa usando pthreads, considerando **N** threads leitoras e **M** threads escritoras. A base de dados compartilhada (região crítica) deverá ser um array, e threads escritoras deverão continuamente (em um laço infinito) escrever no array em qualquer posição. Similarmente, as threads leitoras deverão ler dados (de forma contínua) de qualquer posição do array. As seguintes restrições deverão ser implementadas:

1. As threads leitoras podem simultaneamente acessar a região crítica (array). Ou seja, uma thread leitora não bloqueia outra thread leitora;
2. Threads escritoras precisam ter acesso exclusivo à região crítica. Ou seja, a manipulação deve ser feita usando exclusão mútua. Ao entrar na região crítica, uma thread escritora deverá bloquear todas as outras threads escritoras e threads leitoras que desejarem acessar o recurso compartilhado.

Dica: *Você deverá usar mutex e variáveis de condição.*

4. O Algoritmo de Boruvka é um dos mais antigos algoritmos para obter a árvore geradora mínima de um grafo (*minimum spanning tree*).
Ele consiste em dividir o grafo em subgrafos e calcula as respectivas árvores geradores mínimas. Pode ser considerado uma variação do algoritmo de Kruskal.

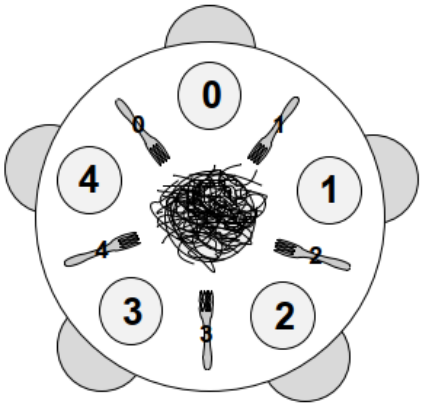
```
ALGORITMO BORUVKA(G)
    Crie uma floresta F com cada nó do grafo
    Crie um vetor de LIDER com todos os nós

    para cada nó u em G
        LIDER[u] = u

    Enquanto F > 1 faça
        para cada componente c em G
            ache a aresta cv de menor custo
            se FIND(c, LIDER) diferente de FIND(v, LIDER) então
                adicione cv a F
                UNION(c, v)
FIM
```

Perceba que pode-se buscar a aresta de menor custo para cada componente de forma concorrente, alocando uma thread por componente.
Faça em implementação em pthreads da versão do algoritmo concorrente

5. O jantar dos filósofos foi proposto por Dijkstra em 1965, e há diversas variações do mesmo problema. O problema básico envolve uma mesa redonda com 5 pratos, 5 garfos e uma tigela de macarrão. 5 filósofos, os quais representam threads que se interagem, executam em um laço:

```
while(TRUE) {
       think();
       get_forks();
       eat();
       put_forks();
}
```

Os garfos representam recursos que as threads precisam obter de forma mutuamente exclusiva para executar. São necessários 2 garfos que estão adjacentes, de acordo com a figura abaixo. Filósofos com fome precisam aguardar a disponibilidade de garfos para prosseguir a execução.



Os filósofos possuem uma variável local i que os identificam entre os valores de 0 a 4. Similarmente, os garfos são numerados de 0 a 4, de tal forma que o filósofo i possui o garfo i a direita e o garfo i + 1 à esquerda..
Você deverá implementar o jantar de filósofos em pthreads que satisfaçam as seguintes condições

- Somente um filósofo pode usar um garfo por vez;
- Deadlock não pode ocorrer;
- Starvation não pode acontecer;
- Mais de um filósofo pode comer ao mesmo tempo;
- Permitir o máximo de concorrência possível;
- Não precisa se preocupar em relação ao tempo para comer, mas o filósofo precisa terminar de se alimentar em algum momento.

6. Uma matriz esparsa é uma matriz em que a maioria de seus elementos tem valor zero. Matrizes desse tipo aparecem frequentemente em aplicações científicas. Ao contrário de uma matriz densa, em que é necessário levar em consideração todos os valores da matriz, em uma matriz esparsa podemos nos aproveitar da estrutura da matriz para acelerar a computação de resultados. Muitas vezes, faz-se mesmo necessário aproveitar essa estrutura, ao se lidar com matrizes esparsas de dimensões imensas (da ordem de 10^6 x 10^6) para que a computação termine em um tempo razoável, visto que a multiplicação de matrizes tradicional tem complexidade $O(n^3)$. Nesta questão, você deverá implementar algoritmos para realizar algumas operações comuns sobre matrizes esparsas de números de ponto-flutuante.

Para auxiliar na definição de uma matriz esparsa, definiremos primeiramente um vetor esparso: Um vetor esparso será dado por um vetor de pares ***(índice,valor)***, no qual o primeiro elemento do par indica o índice de um elemento não-zero do vetor, e o segundo elemento indica seu valor. Portanto, o vetor `{0,0,0,1.0,0,2.0}`, em forma de vetor esparso, será representado por `{Par (3,1.0),Par (5,2.0)}` A implementação dos vetores esparsos fica a cargo do aluno. Uma matriz esparsa será dada por um vetor de vetores esparsos.
Portanto, a matriz esparsa a seguir:

```
2.0   -1.0  0.0   0.0
-1.0  2.0   -1.0  0.0
0.0   -1.0  2.0   -1.0
0.0   0.0   -1.0  2.0
```

Seria representada como:

```
{{(0,2.0),(1,-1.0)},
{(0,-1.0),(1,2.0),(2,-1.0)},
{(1,-1.0),(2,2.0),(3,-1.0)},
{(2,-1.0),(3,2.0)}}
```

O aluno deve implementar as seguintes operações sobre matrizes esparsas:

- Multiplicação de uma matriz esparsa por um vetor denso (vetor comum de C)
- Multiplicação de uma matriz esparsa por outra matriz esparsa
- Multiplicação de uma matriz esparsa por uma matriz densa (matriz comum de C)

Todas essas funções devem usar paralelismo. Ex: na multiplicação, cada linha da matriz deve ser gerenciada por uma thread. Portanto, deve-se ter uma thread para cada linha da matriz. *Em sala de aula, foi visto uma abordagem para múltiplicação de matrizes usando múltiplas threads. Você pode adaptá-la para considerar matrizes esparsas.*

7.Uma matriz quadrada é considerada um quadrado latino se um símbolo (ex: número inteiro) aparece somente uma vez em uma linha ou coluna. Implemente em pthreads um programa que verifica se uma matriz quadrada é um quadrado latino.