Algoritmo aproximativo de TSP

Para realizar o trabalho, foram implementados 3 algoritmos em C++:

- Um algoritmo de força bruta sequencial, que testa todas as alternativas unicas, ou seja, sem repetição.
- Um algoritmo 2-aproximativo usando o algoritmo de Prim para gerar uma árvore de abrangência mínima seguido de uma busca por profundidade.
- Um algoritmo de força bruta paralelo.

Assim como esperado, o algoritmo de força bruta executa em tempo exponencial e rapidamente torna-se intratável, enquanto o algoritmo aproximativo é consistente em tempo de execução, roda em um unico núcleo e consegue resolver até o problema mais difícil ordens de magnitude mais rapidamente que os outros retornando resultados próximos do ótimo.

Para tornar a luta um pouco mais justa, foi implementado o algoritmo de força bruta com execução paralela, utilizando até N threads, onde N = Número de nodos do grafo. Porém, como é possível ver na tabela mais a frente, nao foi de grande ajuda.

Tempos de execução

Os tempos, foram gravados usando um Ryzen 5 3600 rodando Pop!_OS(linux 5.11-rc7) e utilizando o compilador `clang++`.

> O comando exato utilizado foi `CXX=clang++ ./run.sh > logs/(date --iso-8601=seconds).txt` no shell fish.

> **Definições**
>
> BFMT
> : Algoritmo de força bruta paralelo.
>
> BFST
> : Algoritmo de força bruta sequencial.
>
> APPROX
> : Algoritmo 2-aproximativo.
>
> DNF
> : Execução foi pulada por demorar demais.

Melhores tempos de execucao

			BFMT		BFST		APPROX	
Grafo	**Nodos**	**Resposta Esperada**	**Tempo**	**Resposta**	**Tempo**	**Resposta**	**Tempo**	**Resposta**
P1	11	253	5ms[1]	253 *(1x)*	20ms[2]	253 *(1x)*	13us[3]	270 *(~1.07x)*

			BFMT		BFST		APPROX	
Grafo	**Nodos**	**Resposta Esperada**	**Tempo**	**Resposta**	**Tempo**	**Resposta**	**Tempo**	**Resposta**
P2	6	1248	75us[4]	1248 *(1x)*	1us[5]	1248 *(1x)*	2us[6]	1272 *(~1.02x)*
P3	15	1194	73s[7]	1194 *(1x)*	409s[8]	1194 *(1x)*	13us[9]	1519 *(~1.27x)*
P4	22	7013	DNF		DNF		32us[10]	8308 *(~1.18x)*
P5	29	27603	DNF		DNF		54us[11]	35019 *(~1.26x)*

Como rodar o projeto

O sistema de build escolhido foi o mesonbuild, pois é conveniente e esta sendo cada vez mais utilizado é só o que o aluno sabe usar.

Para instalar basta seguir o guia do meson.

Depois, ao rodar `./run.sh`, todo processo de build é executado, incluindo baixar e compilar localmente a dependencia `{fmt}` automaticamente.

> **Flags**
>
> É possível controlar qual algoritmos serão executados passando a flag adequada para o programa, por exemplo:
>
> ```
> ./run.sh --p3:no-seq-brute-force
> ```
>
> Para ver todos as flags leia o main.

1. logs/2021-05-20T01:07:58-03:00.txt, Ciclo = [0, 7, 4, 3, 9, 5, 2, 6, 1, 10, 8, 0] ↩
2. logs/2021-05-20T01:16:24-03:00.txt, Ciclo = [0, 7, 4, 3, 9, 5, 2, 6, 1, 10, 8, 0] ↩
3. logs/2021-05-20T01:45:26-03:00.txt, Ciclo = [0, 7, 2, 10, 1, 6, 4, 3, 5, 9, 8, 0] ↩
4. logs/2021-05-20T00:53:57-03:00.txt, Ciclo = [0, 5, 4, 3, 2, 1, 0] ↩
5. logs/2021-05-20T00:53:57-03:00.txt, Ciclo = [0, 5, 4, 3, 2, 1, 0] ↩
6. logs/2021-05-20T00:53:57-03:00.txt, Ciclo = [0, 1, 5, 4, 3, 2, 0] ↩
7. logs/2021-05-20T01:34:56-03:00.txt, Ciclo = [0, 1, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 0] ↩
8. logs/2021-05-20T01:07:58-03:00.txt, Ciclo = [0, 1, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 0] ↩
9. logs/2021-05-20T00:53:57-03:00.txt, Ciclo = [0, 1, 2, 3, 4, 8, 7, 5, 6, 14, 13, 12, 11, 10, 9, 0] ↩
10. logs/2021-05-20T00:53:57-03:00.txt, Ciclo = [0, 7, 21, 3, 17, 16, 1, 2, 15, 12, 11, 6, 5, 19, 18, 9, 8, 10, 20, 13, 14, 4, 0] ↩

11. logs/2021-05-20T00:53:57-03:00.txt, Ciclo = [0, 1, 5, 4, 3, 6, 2, 8, 7, 11, 9, 10, 12, 13, 16, 17, 18, 14, 21, 22, 20, 28, 27, 25, 19, 15, 24, 26, 23, 0] ↩