Sistemas Operativos
2024 / 2025

Licenciatura em Engenharia Informática

Trabalho Prático 1

Introdução

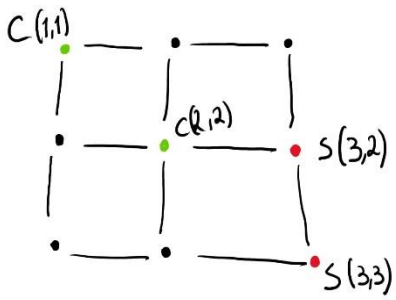
Por causa do COVID-19, o Eng. José Covídeo foi encarregue de projetar um sistema que permita aos cidadãos de *Manhattan* (*New York*) deslocar-se a um supermercado sem se cruzarem com nenhum outro cidadão. Estão disponíveis no mapa da cidade, as localizações dos supermercados, que se situam todos em esquinas e as moradas dos cidadãos que, para este efeito, se situam também nas esquinas.

Tratando-se de *Manhatan*, as ruas têm um arranjo em quadriculado absolutamente regular, e considera-se que em todas as ruas se circula em ambos os sentidos. As avenidas estão numeradas de 1 a M e são na direção NORTE-SUL, enquanto as ruas estão numeradas de 1 a N e são na direção ESTE-OESTE. Os cruzamentos são definidos por um par de números, sendo que o par (A,B) corresponde ao cruzamento da Avenida A com a rua B.

Dados um conjunto de supermercados abertos e de cidadãos que querem fazer compras a uma dada hora, o sistema deverá determinar qual o número máximo de cidadãos que pode deslocar-se a um supermercado, sem correr o risco de se encontrar com outro cidadão, numa rua, avenida ou cruzamento, inicial, intermédio ou final do seu percurso.

Para simplificar iremos considerar que não podem existir dois supermercados na mesma esquina. Iremos também considerar que não podem morar dois cidadãos no mesmo cruzamento. Caso existam dois supermercados ou dois cidadãos na mesma esquina/cruzamento só um pode ser usado na solução (devido a risco de contágio).

Exemplo:



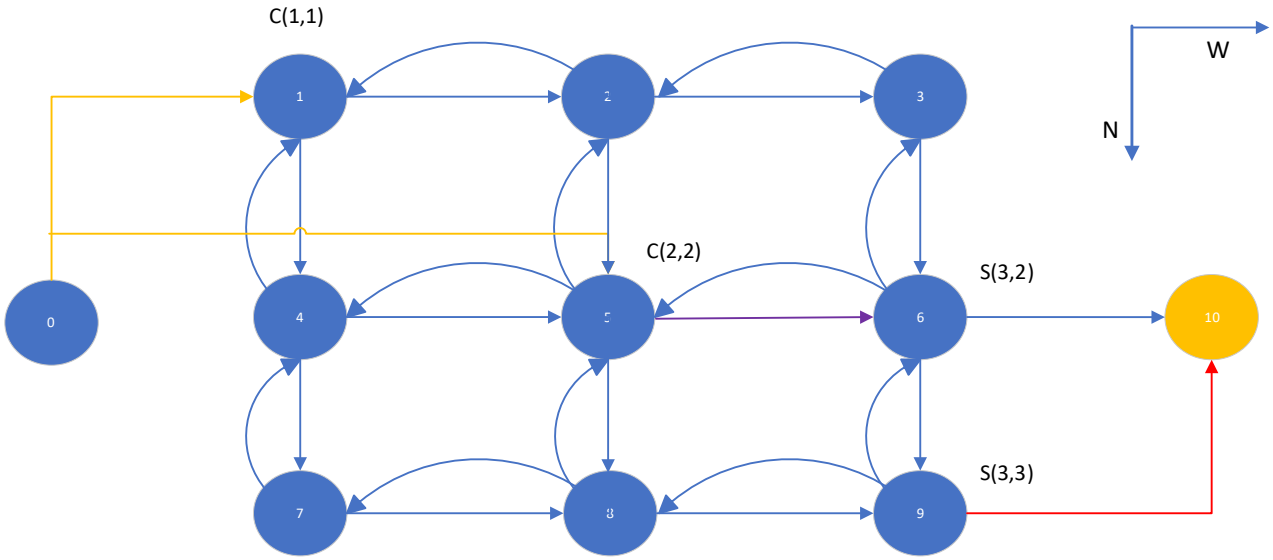


Figura 1

Neste exemplo temos:

- número de avenidas (M) igual a 3
- número de ruas (N) igual a 3
- número de supermercados é igual a 2
- número de cidadãos é 2.

As coordenadas dos supermercados são: (3, 2) e (3, 3)

As coordenadas dos cidadãos são: (1, 1) e (2, 2).

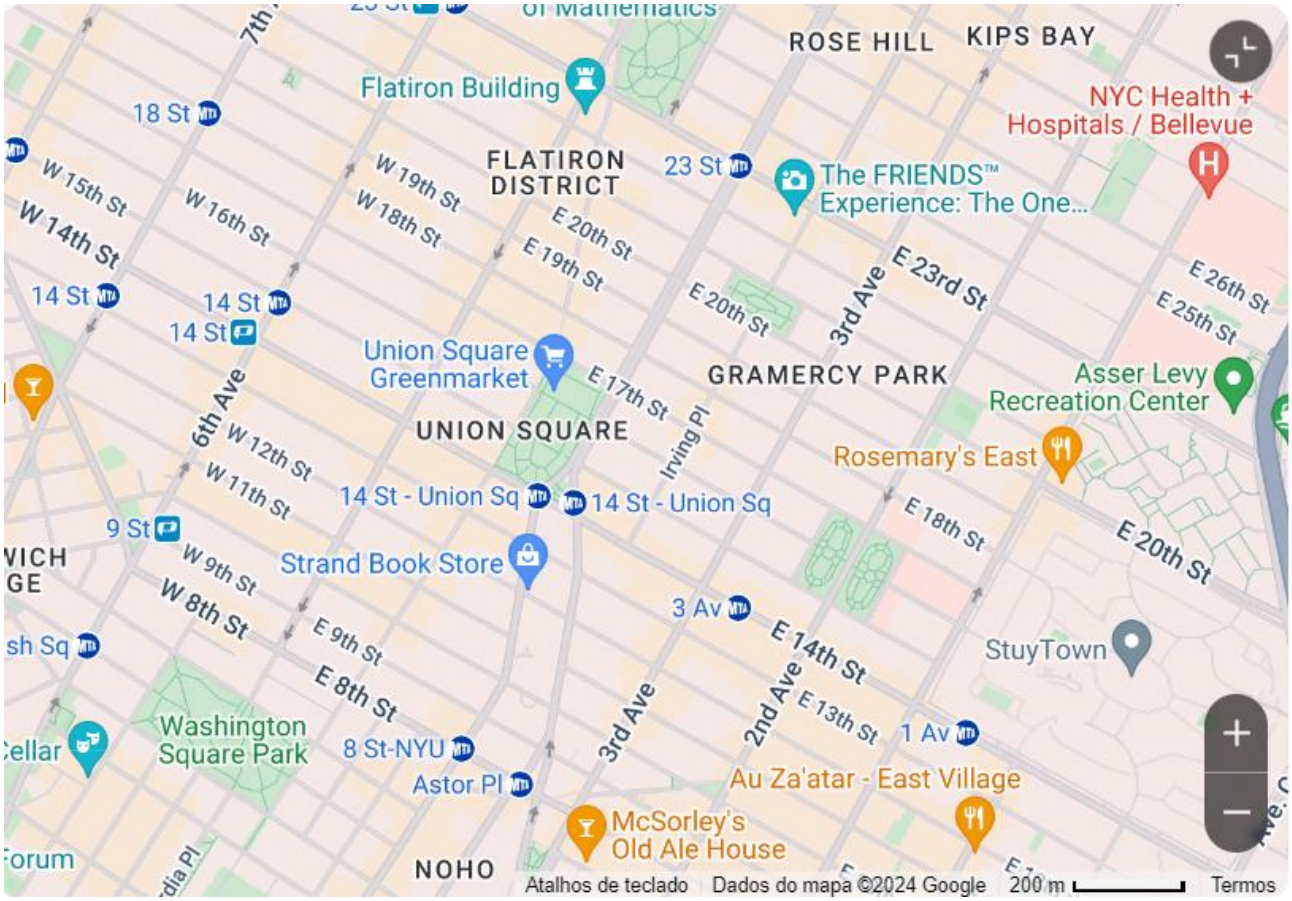
Na figura 1 encontra-se uma representação possível para este problema, usando um grafo com 11 vértices e vários arcos conectando os vértices. Nesta representação cada vértice representa um cruzamento, e cada arco o caminho possível a partir de cada vértice.

O número total de vértices NT = M * N + 2.

Os supermercados são representados pelos arcos que terminam no último vértice (v_{10}), no exemplo anterior os arcos (6,10) e (9,10).

Os arcos que começam no vértice v_0 representam os cidadãos, neste caso (0,1) e (0,5).

As avenidas são numeradas da esquerda para a direita (E-W) e correspondem às colunas no grafo, e as ruas são numeradas de cima para baixo (S-N) e correspondem às linhas.



Mapa de parte de *New York* (*Manhatan*), retirado do Google Maps

Algoritmos de resolução

A algoritmo mais óbvio é de tentar que cada cidadão alcance um supermercado, testando todos os caminhos possíveis. Para isso deve evitar caminhos que já tenham sido visitados por outros cidadãos. Este algoritmo tem um custo computacional proibitivo quando o número de vértices (cruzamentos) aumenta.

O algoritmo proposto para este trabalho é o algoritmo **Find Safe Citizens (FSCS)**, que procura soluções para vários cidadãos, com o seguinte funcionamento:

1. Calcula o **número máximo de soluções** possível, como sendo o mínimo entre o número de supermercados e o número de cidadãos.
2. Inicializa a **lista de Soluções** (lista de caminhos válidos), **lista de nós visitados**, e o **número de soluções.**
3. Invoca **Find Safe Citizen** e guarda o resultado numa variável **result**.
4. Se **result** for diferente de zero incrementa **número de soluções**
5. Se **result** for diferente de 0 e **máximo de soluções** for diferente de **número de soluções,** segue para o passo 3.
6. Devolve o **número de soluções** e a **lista de soluções**.

O algoritmo **Find Safe Citizen**, procura uma solução para um cidadão e funciona da seguinte forma:

1. Inicializa a **lista dos próximos nós** como sendo a lista de sucessores do nó inicial (nó 0).
2. Se a **lista dos próximos nós** for vazia, sai e devolve 0.
3. Escolhe aleatoriamente um nó da **lista dos próximos nós**.
4. Se esse nó for um supermercado junta-o à **lista de nós visitados por este cidadão,** atualiza **a lista de soluções** e segue para o passo 8.
5. Se esse nó ainda não foi visitado, atualiza a **lista de nós visitados por este cidadão**, a **lista de nós visitados**, e a **lista dos próximos nós**.
6. Se o nó já foi visitado remove-o da **lista dos próximos nós**.
7. Se a **lista dos próximos nós** não for vazia segue para o passo 3,
8. Se foi encontrada um supermercado, remove todos os arcos que terminem em nós da **lista de nós visitados por este cidadão.**

9. Devolve **um** se foi encontrado um supermercado e **zero** em caso contrário.

No fim, o algoritmo deverá ser capaz de retornar uma solução que inclui o número de cidadãos que conseguiram chegar em segurança a um supermercado, assim como o caminho (lista de vértices) que cada cidadão percorre desde sua casa até ao supermercado.
Note que a melhor solução encontrada pelo algoritmo pode ou não ser a melhor solução em termos globais.

Implementação concorrencial do algoritmo FSCS

Dado que o algoritmo FSCS tem uma forte componente aleatória, um dos grandes fatores que pode influenciar a solução é o número de iterações realizadas pelo algoritmo (ou de forma indireta, o tempo que se dá ao algoritmo para tentar encontrar a melhor solução).

Desta forma, propomos a implementação paralela e concorrencial do algoritmo nas suas versões *Base, Avançada* e *Original*.

Versão Base

1. Criar *m* processos (número parametrizável) em que cada processo corre o algoritmo FSC.
2. À medida que cada processo encontra uma solução melhor que a anterior, coloca a sua solução na memória partilhada.
3. Dado que dois ou mais processos podem aceder simultaneamente à memória partilhada e corrompê-la, a atualização desta deve ser feita de forma controlada usando mecanismos de sincronização como os semáforos.
4. Ao fim de algum tempo, termina-se a execução e informa-se o utilizador da melhor solução encontrada.

Versão Avançada

Igual à versão base, exceto que após o passo 2:

a) Cada um dos processos é parado e reiniciado com uma solução parcial.
b) A solução parcial corresponde a uma cópia parcial da melhor solução encontrada, do total de caminhos encontrados (cada uma sendo uma lista de nós percorridos por um cidadão até ao supermercado) são escolhidos **m** caminhos aleatoriamente ($\boldsymbol{m} \in [0, tamanho\ soluções/2]$).
c) Depois de cada processo copiar os caminhos aleatórios ele irá recomeçar o algoritmo FSC a partir desse ponto.

Versão Original

Outra versão que resolva o problema em tempo útil e que seja objetivamente melhor que as anteriores.

Desenvolvimento

A aplicação deverá ser feita na linguagem de programação C, em Linux, usando as técnicas de programação concorrencial utilizadas nas aulas laboratoriais, nomeadamente *fork*, funções de manipulação de memória partilhada, semáforos, etc.

Input

- O ficheiro de entrada contém informação sobre o número de ruas e avenidas, bem como sobre o número e morada de cidadãos e supermercados. O input é definido da seguinte forma:
- Uma linha com o número de avenidas e o número de ruas, M e N;
- Uma linha com o número de supermercados e o número de cidadãos, S e C;
- S linhas, cada uma com a localização de cada supermercado, definida pelo número da avenida e pelo número da rua do respetivo cruzamento;
- C linhas, cada uma com a morada de cada cidadão, definida pelo número da avenida e pelo número da rua do respetivo cruzamento.

O programa deverá ser capaz de ser invocado pela linha de comandos passando como argumentos o nome do ficheiro de texto com o problema, o número de processos a serem criados e o tempo máximo de execução do algoritmo (em milisegundos). Por exemplo, o comando **safe instances_2.txt 4 50** deverá executar o ficheiro de teste "instances_2.txt" usando 4 processos em paralelo durante 50 milisegundos.

Resultados

O output do programa deverá indicar um inteiro representando o número máximo de cidadãos que podem deslocar-se aos supermercados sem se encontrarem.

De modo a se validar a qualidade do algoritmo, deverá ser construída uma tabela com as seguintes colunas:

a) Número do teste (de 1 a 10).
b) Nome do teste.
c) Tempo total de execução (ms).
d) Número de processos usado (parâmetro p na descrição dos algoritmos).
e) Número de cidadãos podem deslocar-se aos supermercados sem se encontrarem
f) Lista de caminhos que representa os caminhos de cada um dos cidadãos "safos"
g) Número de iterações necessárias para chegar à solução encontrada.
h) Tempo que demorou até o programa atingir a solução encontrada (ms).

Cada teste deverá ser repetido 10 vezes para os mesmos parâmetros de entrada, e deverá ser possível obter valores médios de tempo e número de iterações, assim como o número de vezes em que se encontrou a solução ótima.
Os ficheiros de teste a utilizar serão disponibilizados no Moodle da disciplina, assim como um ficheiro com alguns resultados.

Entrega e avaliação

Os trabalhos deverão ser realizados em grupos de 2 ou 3 alunos da mesma turma de laboratório, e deverão ser originais. Trabalhos plagiados ou cujo código tenha sido partilhado com outros serão atribuídos nota **zero**.

Todos os ficheiros deverão ser colocados num **ficheiro zip** (com o número de todos os elementos do grupo) e submetido via *Moodle* **até às 8:30 do dia 2/Dezembro/2024**. Deverá também ser colocado no zip um **relatório em pdf** com a identificação dos alunos, as tabelas de resultados, a percentagem de participação de cada aluno no projeto e as descrições das soluções que considerarem relevantes. Este documento deverá ser mantido curto e direto (2-3 páginas).

Irá considerar-se a seguinte grelha de avaliação:

Algoritmo FSCS	04 val.
Algoritmo concorrencial	
Versão Base	04 val.
Versão Avançada	03 val.
Outra versão original	02 val.
Utilização de memória partilhada	01 val.
Utilização de semáforos	01 val.
Relatório com a tabela de testes	02 val.
Qualidade da solução e código	03 val.

As discussões dos projetos serão realizadas na semana seguinte à entrega do projeto, no horário das aulas laboratoriais. As notas poderão ser atribuídas aos alunos de forma individual.

Bom trabalho!