Relatório 2º projecto ASA 2019/2020

Grupo: tp032

Alunas: Beatriz Venceslau (93734) e Carolina Ramos (93694)

Descrição do Problema e da Solução

O problema passa por descobrir quantos supermercados podem ser acedidos, sem que os cidadãos se cruzem durante o caminho para o supermercado. Para isto é nos dado o número de ruas e avenidas, que formam um grafo de uma cidade, em que os cidadãos e os supermercados se encontram nas esquinas (vértices do grafo).

A localização dos cidadãos e dos supermercados abertos é nos dada como coordenadas (avenida, rua). Se houver mais que um supermercado num cruzamento, apenas um poderá ser acedido.

A nossa solução passa por primeiro converter cada cruzamento do grafo em vértice, através da fórmula: Rua + (Avenida - 1) * NumTotalRuas. Um vértice é composto pelo seu identificador vIn, e uma lista de até 5 vizinhos vOut; pela capacidade da aresta entre vIn e vOut e pelo flow. Assim uma vez que vIn está ligada a diversos vizinhos, todos os vizinhos conseguem verificar se vIn já foi acedido ou não, através da diferença entre a capacidade e flow. O grafo é construído com o uma super sourse (vértice 0) que aponta para todos os cidadãos, e com um super target (vértice numCruzamentos+1) que recebe todos os vértices dos supermercados.

Para calcular o número máximo de caminhos possíveis, a nossa solução utiliza o algoritmo Ford-Fulkerson. Este usa a BFS para encontrar o caminho mais curto entre um cidadão e um supermercado, sem se cruzar com outro cidadão. De seguida, se este caminho existir, o flow das arestas visitadas é incrementado em 1. No fim, quando a BFS não encontrar mais nenhum caminho, o algoritmo Ford-Fulkerson retorna o flow do vértice super target, que representa o max flow.

Análise Teórica

Considerando M = avenidas, N = ruas, V=M*N=n+1 e G = (V,E).

- <u>Leitura dos dados de entrada: simples leitura do input, com ciclos lineares.</u>

```
for i = 0 to n+1 do
    lista[i].vIn = i;
    lista[i].flow = 0;
    lista[i].capacity = 1;
    for j = 0 to 4 do
        lista[i].vOut[j] = 0;
    end for
end for
for i = 0 to supermercados do
    scanf("%d %d",&avenida, &rua);
    Vert = calculaVertice();
    lista[vert].vOut[0] = n + 1;
end for
```

```
for i = 0 to cidadaos do
    scanf("%d %d", &avenida, &rua);
    vert = calculaVertice();
    lista[0].vOut[i] = vert;
end for
```

Supermercados = S e cidadãos = C
Logo Θ(5V + S + C) = O(V)

- Processamento do grafo para fazer alguma coisa.

```
for i = 1 to n do
    if i % numRuas != 1 do          /*tem vizinho para cima*/
        d = i - 1;
        for j = 0 to lista[i].vOut[j] != 0 do
            lista[i].vOut[j] = d;
        end for
    end if

    if i > numRuas do               /*tem vizinho a esquerda*/
        d = i - numRuas;
        for j=0 to lista[i].vOut[j] != 0 do
            lista[i].vOut[j] = d;
        end for
    end if

    if i <= n - numRuas do          /*tem vizinho a direita*/
        d = i + numRuas;
        for j=0 to lista[i].vOut[j] != 0 do
            lista[i].vOut[j] = d;
        end for
    end if

    if i % numRuas != 0 do          /*tem vizinho para baixo*/
        d = i + 1;
        for j=0 to lista[i].vOut[j] != 0 do
            lista[i].vOut[j] = d;
        end for
    end if
```

Cada ciclo for dentro dos if realiza-se no pior dos casos 5 vezes.
Um vértice pode, no pior dos casos entrar em todos os 4 ifs.
E o for corre V vezes. Logo, O(20 * (V-1)) = O(V).

Relatório 2º projecto ASA 2019/2020

Grupo: tp032

Alunas: Beatriz Venceslau (93734) e Carolina Ramos (93694)

- Aplicação do algoritmo bfs para fazer algo. Logo, O(V+E)

```
for u = 0 to n+1 do
    color[u] = WHITE;
end for
enqueue();
while head!=tail do
    u = dequeue();
    if u == 0 do
       for v = 0 to cidadaos do
          indVizinho = lista[u].vOut[v];
          if (color[indVizinho]==WHITE && lista[indVizinho].capacity -
lista[indVizinho].flow > 0) do
             enqueue();
             pred[indVizinho] = u;
           end if
       end for
    end if
    else do
      for v = 0 to 5 do
       indVizinho = lista[u].vOut[v];
       if (indVizinho == n+1 && lista[indVizinho].capacity -
lista[indVizinho].flow > 0) do
          pred[indVizinho] = u;
          return 1;
       end if
       else if (color[indVizinho]==WHITE && lista[indVizinho].capacity -
lista[indVizinho].flow > 0) do
          enqueue();
          pred[indVizinho] = u;
          if (lista[indVizinho].vOut[2] == 0) do
             if (lista[indVizinho].vOut[0] != u)do
                condicao();
             end if
             else if (lista[indVizinho].vOut[1] != u) do
                condicao();
             end else
     end for
end while
```

- Aplicação do algoritmo Ford-Fulkerson para fazer algo. Logo, $O(V*E^2)$

```
while bfs() do
    for u = n+1 to 0 do
        u = pred[u];
        lista[u].flow += 1;
     end for
end while
```

Relatório 2º projecto ASA 2019/2020

Grupo: tp032

Alunas: Beatriz Venceslau (93734) e Carolina Ramos (93694)

- Transformação dos dados com uma dada finalidade. O(1)

  rua + (avenida - 1) * numRuas;

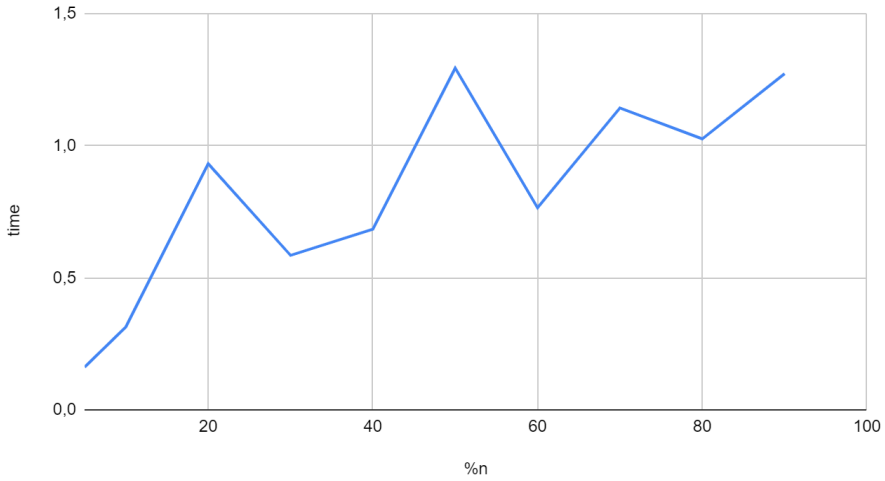
- Apresentação dos dados. O(1)

  printf("%d\n", max_flow(s,t, lista));

Complexidade global da solução: $O(V*E^2)$

Avaliação Experimental dos Resultados

Foram gerados vários testes em que se fixou o valor das avenidas e das ruas a 100, variando o número de cidadãos e supermercados, sendo estes equivalentes a uma percentagem do número de vértices (10000).
Exemplo: n = 10000, cidadãos = 2000, supermercados = 2000, equivale no gráfico a %n = 20%.



Podemos concluir que o gráfico gerado não está de acordo com a análise teórica, uma vez que o programa em alguns casos dá resposta errada, pois não obtivemos cotação máxima no Mooshak.

Se não fossem esses problemas, o programa teria um gráfico quadrático, que corresponderia à análise teórica dos algoritmos.

Referências:

Material disponível na página da cadeira: (Power Points)
-Análise e Síntese de Algoritmos Edmonds-Karp.
-Análise e Síntese de Algoritmos BFS. Caminhos mais curtos.
Webgrafia:
-https://sahebg.github.io/computersceince/Maximux-flow-ford-fulkarson-algorithm-c-program-example/