U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Tema 6 – Supermercado ao Domicílio

Turma 2MIEIC01 - Grupo A

Tiago Lascasas dos Santos - up201503616

Leonardo Gomes Capozzi - up201503708

Ricardo Miguel Oliveira Rodrigues de Carvalho – up201503717

Data: 7/4/2017

Índice

Descrição do Tema

O tema deste projeto, “Supermercado ao Domicílio”, tem como objetivo a criação de um sistema que permita a uma cadeia de supermercados gerir as entregas ao domicílio de compras feitas pela internet.

Como forma de representar um mapa da zona de aplicação deste sistema, foi usado um grafo gerado a partir de ficheiros de texto criados com o programa *OSM2TXT Parser* que recebe mapas no formato *OSM XML* obtidos a partir do www.openstreetmap.org.

Por ser uma cadeia de supermercados, vários membros da cadeia podem fazer entregas, sendo, portanto, necessário avaliar várias hipóteses de forma a otimizar a distância e o tempo de viagem de cada camião de entrega. Há ainda a hipótese de um cliente ser inacessível a partir de qualquer mercado, o que requer uma análise de conectividade entre os mercados e os clientes.

Aquando da conclusão do projeto deverá ser possível, para uma dada lista de mercados e clientes, fazer uma distribuição das entregas, partindo de qualquer mercado ou de um mercado específico para qualquer cliente de forma a permitir que, de forma automática ou manual, haja dados que permitam escolher o mercado de onde sai o camião de entregas de forma a minimizar o tempo e distância de cada deslocação.

Descrição das Soluções

A partir do enunciado, foram retirados e isolados cinco problemas distintos, os quais se encontram seguidamente detalhados.

De modo a uniformizar a notação de cada descrição, procedeu-se às seguintes definições:

- G: grafo com informação sobre a rede viária em causa.
- V: conjunto de todos os nós de G.
- E: conjunto de todas as arestas de G.
- M: conjunto de todos os mercados, $M \subset V$.
- P: conjunto de todos os clientes, $P \subset V$.
- Vm: velocidade média de um camião
- $d(v1, v2)$: distância do vértice v1 ao vértice v2, tal que $v1, v2 \in V$.
- $t(Vm, d, n)$: tempo decorrido para percorrer a distância d à velocidade média Vm, passando por n clientes. Por cada cliente, um factor tc é somado ao resultado, factor este referente ao tempo que um camião demora a parar, entregar a encomenda ao cliente e voltar a andar.

1. Estudo de Conectividade

De modo a determinar se um cliente pode ou não ser atendido por um determinado mercado, procedeu-se a um estudo de conectividade baseado num algoritmo de Pesquisa em Profundidade (DFS):

- Input - M, P
- Para cada mercado $m: m \in M$, realiza-se uma DFS desse mercado para todos os outros nós $v: v \in V$.
- Para cada vértice $v: v \in V$ atingível por uma DFS a partir de m, se $v \in P$, v fica marcado como atingível por m.
- Output: nenhum; cada vértice de P fica marcado como pertencendo ou não a cada um dos mercados de M.
- Complexidade temporal: $|M| \cdot O(|V| + |E|)$ no pior caso (todos os vértices de V serem atingíveis por cada mercado de M)
- Complexidade espacial: $O(|V|)$ no pior dos casos (cada vértice é visitado apenas uma vez)

Na prática, o algoritmo implementado fica sempre longe de atingir os piores casos, visto que após finalizada a visita a começar em m, não é realizada qualquer visita a partir dos nós não explorados, como numa pesquisa DFS normal.

2. Entrega a partir de um mercado para um cliente

De modo a determinar qual a distância e o tempo de entrega de um mercado para um cliente, procedeu-se a uma implementação modificada do algoritmo de Dijkstra:

- Input - $m: m \in M, p: p \in P$
- A modificação introduzida no algoritmo de Dijkstra consiste na paragem do algoritmo assim que o vértice p seja alcançado, retornando, então, $d(m,p)$.
- A partir da distância d retornada pelo algoritmo, pode-se calcular $t(d, Vm, 1)$, obtendo-se assim, também, o tempo que o camião demora a chegar do mercado ao cliente.
- Output: $t(d, Vm, 1)$
- Complexidade: $O(|E| \cdot log|V|)$ no pior dos casos (vértice p está o mais longe possível de m).

Na prática, o pior caso é raramente atingível, porque na maior parte dos casos o algoritmo para muito antes de atingir todos os vértices do grafo.

3. Entrega a partir de todos os mercados para um cliente

Este problema é semelhante ao anterior, mas em vez de se especificar qual o mercado a partir do qual se quer chegar ao cliente, calcula-se a distância para o cliente a partir de todos os mercados.

- Input – $M, p: p \in P$
- Para cada mercado $m: m \in M$, é corrido o algoritmo de Dijkstra de m para p, obtendo-se, portanto, um conjunto de distâncias $D = \{d(m1, p), d(m2, p), \dots, d(mn, p)\}: m1, m2, \dots, mn \in M$.
- De seguida, são calculados os vários tempos $T = \{t(d1, Vm, 1), t(d2, Vm, 1), \dots, t(dn, Vm, 1)\}: d1, d2, \dots, dn \in D$
- Output: T
- Complexidade: $O(|M| \cdot |E| \cdot \log|V|)$ no pior dos casos (vértice p está o mais longe possível de todos os mercados m).

Tal como o algoritmo anterior, o pior caso é raramente atingível, visto que o algoritmo geralmente para antes de atingir todos os nós.

4. Entrega de um mercado para todos os clientes

Para este problema foi inicialmente idealizada a aplicação do algoritmo de Prim, de modo a determinar uma árvore de expansão mínima (MST) que daria o caminho mínimo que conectava o mercado a cada um dos clientes. Contudo, visto que G é dirigido e que tanto o algoritmo de Prim como o de Kruskal são aplicáveis apenas a grafos não dirigidos, não foi possível abordar o problema por esta via. O problema poderia ter sido possivelmente resolvido recorrendo a caminhos e circuitos de Euler, mas visto que estes estão fora do escopo do trabalho também não abordámos o problema por essa perpsectiva. Portanto,

decidimos calcular vários caminhos entre um mercado e todos os seus clientes através do algoritmo de Dijkstra, de modo a atender o maior número de clientes por caminho. Depois, consoante o número de camiões disponível, é efectuada uma divisão destes caminhos pelos vários camiões, de modo a minimizar o tempo de entrega das compras dos clientes. Esta perspectiva não inclui o tempo de retorno de cada camião, sendo que na prática se pode estimar ser o mesmo que o caminho de ida. Procedeu-se, então, à seguinte divisão por partes:

- Input inicial: $m: m \in M, P$
- **1ª parte** – é criado um conjunto que contém apenas os clientes que estejam marcados como atingíveis pelo mercado em questão.
    - Input: $P, m: m \in M$
    - Output: Pv (conjunto de todos os $p: p \in P$ que são atingíveis por m)
    - Complexidade: $O(|P|)$
- **2ª parte** – é corrido o algoritmo de Dijkstra a partir de m, calculando a distância de todos os vértices de V até m, com complexidade $O(|E| \cdot log|V|)$.
    - De seguida, calcula-se os caminhos com base no seguinte algoritmo:
        - enquanto $Pv. > 0$
        - Calcular um caminho:
            - Percorrer a lista de vértices V e determinar qual o vértice $pv \in Pv$ que tem maior distância a m, removendo-o de Pv;
            - A partir desse vértice pv, percorrer o caminho de regresso a m, e se passar por um vértice pertencente a Pv, remove-o de Pv;
            - Obtém-se, assim, um conjunto de vértices S ∈ V que indicam o caminho de m a pv, e qual o seu comprimento: $d(m, pv)$,
            - De seguida, calcula-se $t(d, Vm, np), np$ =nº de vértices removidos de Pv;
            - Adiciona-se S a um conjunto de caminhos W;
    - Complexidade: $O(|Pv| \cdot (|V| + |S|))$
- **3ª parte** - finalmente, faz-se o estudo dos camiões:
    - Input: W, n = número de camiões disponível
    - Percorre-se o conjunto W sequencialmente, sendo que este se encontra ordenado do caminho mais longo para o mais curto (consequência do passo anterior).

- Para um caminho w ∈ W, percorre-se um conjunto T de camiões ($|T| = n$), e determina-se qual o camião $t \in T$ cuja soma das distâncias dos caminhos que vai percorrer é mínima, adicionando-se o caminho w a t.
- Assumindo que todos os camiões partem ao mesmo tempo e que o tempo de retorno ou é igual ao de ida ou é desprezável, o tempo máximo que se demora a entregar todas as encomendas é o tempo do camião cuja soma dos caminhos seja máxima. Esta informação, juntamente com a divisão dos caminhos por cada camião, constitui o output do problema.
- Complexidade: $O(|W| \cdot |T|)$

De modo a simplificar a explicação do próximo problema, definir-se-á uma função $paths(Pv, n)$, em que Pv é um conjunto de clientes válidos e n o número de camiões e cujo objetivo é o cálculo dos caminhos e a sua divisão entre os vários camiões, ou seja, pura e simplesmente a junção dos passos 2 e 3. A complexidade desta função é, no pior dos casos (um caminho por cada cliente), $O(|Pv| \cdot |Pv| \cdot (|V| + |S|) + |W| \cdot |T|) + |E| \cdot log|V|)$.

A complexidade final desta solução é, então, $O(|P| + O(path))$.

5. Entregas de todos os mercados para todos os clientes

Este último problema é semelhante ao anterior, só que primeiro agrupa todos os clientes mais próximos de cada mercado, e corre a função paths previamente definida para cada um desses conjuntos.

Input: M, P

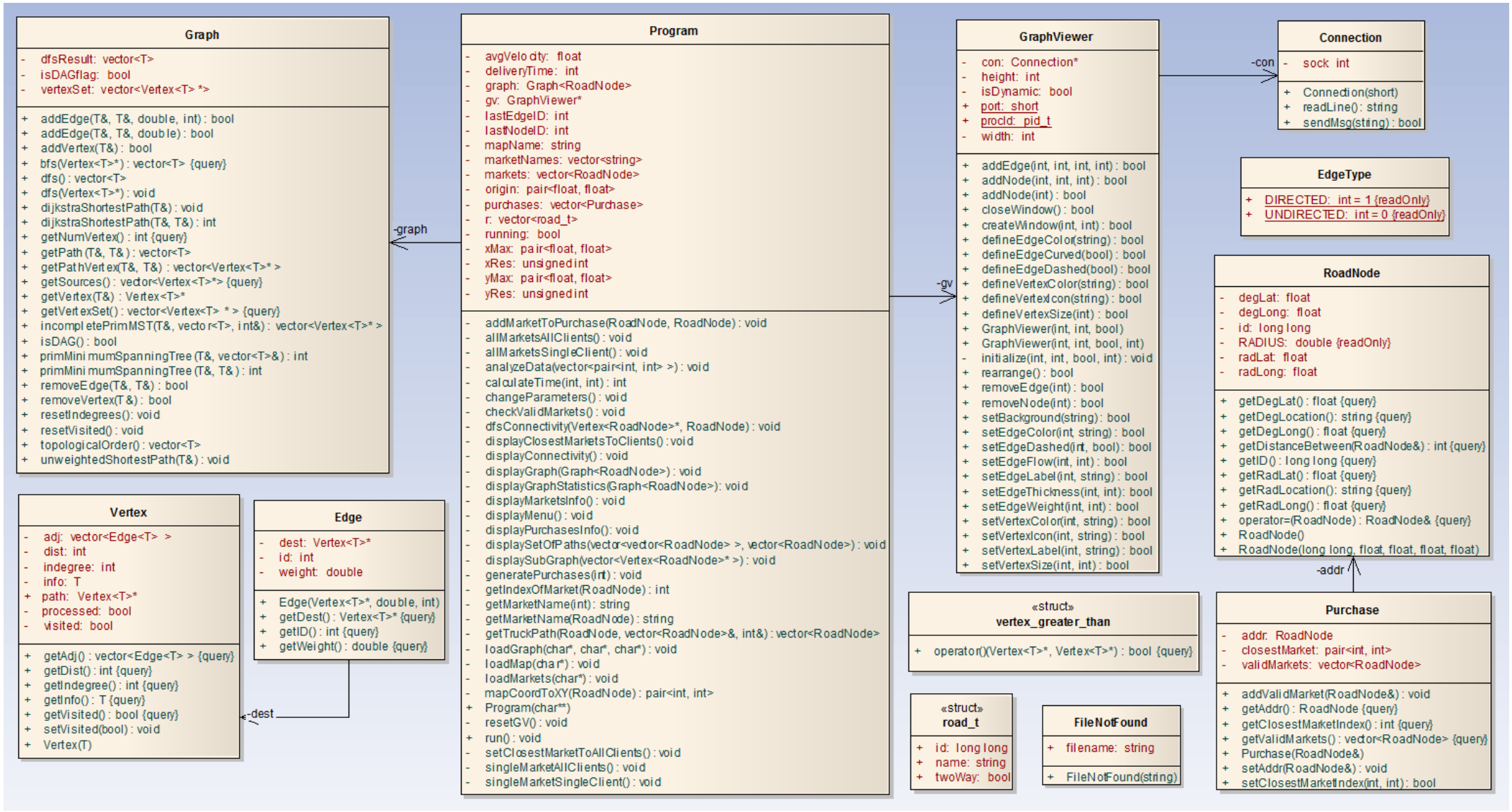
- **1ª parte** – cálculo dos mercados mais próximos de cada cliente pelo algoritmo de Dijkstra:
  - Para cada $m: m \in M$, corre-se o algoritmo de Dijkstra a partir de m para todos os $v: v \in V$.
  - É criado um conjunto Pv para cada m, e cada cliente $p: p \in P$ é adicionado ao conjunto Pv do mercado cuja aplicação do algoritmo de Dijkstra resultou numa distância menor. Cada conjunto Pv pertence ao conjunto Z, |Z| = |M|.
- **2ª parte** - para cada $Pv: Pv \in Z$, é aplicada a função $path(Pv, n)$, em que n é o número de camiões do mercado a que Pv se refere.
- Output: divisão dos caminhos pelos camiões de cada mercado de modo a cada mercado atender os clientes mais próximos.
- Complexidade: $O(|M| \cdot |P| \cdot |E| \cdot log\ |V| + O(path))$.

Considerações sobre a implementação prática das soluções

A implementação das soluções segue as descrições acima apresentadas, sendo que poderá ter sido acrescentada complexidade extra, principalmente para fins de output e apresentação de informação. Contudo, consideramos que o overhead extra introduzido nestes detalhes é desprezável quando comparado com as soluções em si.

É também de notar que a distância física entre dois vértices (ou seja, o peso da aresta que os liga) foi calculada a partir das coordenadas geográficas de cada vértice a partir da fórmula de haversine.

Diagrama de Classes



Lista de Casos de Utilização

1. Mostrar o Grafo
    - Permite exibir, utilizando o *Graph Viewer*, todo o conteúdo atual do grafo (vértices e arestas).

2. Mostrar todos os supermercados
    - Descreve, na consola, todos os supermercados do grafo indicando, para cada um, um id, o nó em que se encontra (numero do nó e respetivas coordenadas geográficas em graus) e o nome.

3. Mostrar todos os clientes/compras
    - Descreve, de forma análoga aos supermercados, todos os clientes, apresentando, em vez do nome, os mercados válidos para cada cliente.

4. Gerar clientes/compras aleatórias
    - Permite gerar aleatoriamente a quantidade pretendida de compras, substituindo a lista de clientes anterior. Por defeito, o programa gera 15 compras quando é inicalizado.

5. Verificar ligação entre todos os clientes e todos os supermercados
    - Apresenta todos os mercados válidos para cada cliente, escrevendo na consola a correspondência entre número da compra e id dos mercados válidos. Neste módulo é utilizada a solução de conectividade anteriormente descrita.

6. Distribuir de um supermercado a um cliente
    - Permite a escolha de uma compra e de um mercado para realizar a entrega, devolvendo: na consola, a distância mais curta a percorrer para fazer a entrega (em metros e em quilómetros) bem como o tempo estimado para a entrega; no Graph Viewer, os nós e arestas que formam o percurso da loja até ao cliente. Nesta funcionalidade foi utilizada a solução para o problema “Entrega a partir de um mercado para um cliente”, descrita anteriormente (página 5).

7. Distribuir de todos os supermercados a um cliente
    - Permite a escolha de uma compra e analisa a entrega a partir de todos os supermercados, devolvendo na consola, para cada supermercado, o id e nome do supermercado, a distância mais curta ao cliente (em metros e em quilómetros) bem como o tempo estimado para a entrega. Este problema foi resolvido implementando a solução anteriormente descrita “Entrega a partir de todos os mercados para um cliente” (página 5).

8. Distribuir de um supermercado a todos os clientes
    - Permite selecionar uma loja para distribuir todas as compras. Neste processo são criados e apresentados no *Graph Viewer* e na consola vários caminhos (um para cada cliente) com a respetiva distancia e tempo de percurso, no entanto, há ainda a hipótese de selecionar o número de camiões disponíveis para fazer esta entrega, diminuindo o número de entregas por camião e, por consequente, o tempo gasto e a distância percorrida. Depois da seleção do número de camiões, as encomendas são distribuídas entre eles de forma a minimizar a distância percorrida por cada um. Para a resolução desta funcionalidade foi implementada a solução "Entrega de um mercado para todos os clientes", anteriormente descrita (página 5).

9. Distribuir de todos os supermercados a todos os clientes
    - Funcionalidade que, dados todos os mercados e todas as compras, permite fazer as entregas a partir do mercado mais próximo a cada cliente, otimizando a distância percorrida para cada entrega. Este módulo foi conseguido pela implementação da solução "Entregas de todos os mercados para todos os clientes" como descrita na secção Descrição da Solução (página 7).

10. Mudar parâmetros de entrega
    - Possibilita a mudança do valor da velocidade média e do tempo que leva cada entrega, sendo que estes valores são inicializados por defeito como 30 km/h para a velocidade média e 2 minutos para o tempo de entrega.

Principais Dificuldades

No decorrer do trabalho, deparámo-nos com dois problemas principais: o primeiro prendeu-se com o facto já mencionado de não ser possível usar uma Árvore de Expansão Mínima, o que nos levou a reavaliar grande parte do projeto e a adotar uma abordagem totalmente nova sobre o problema. O segundo problema deveu-se à API do GraphViewer facultada: sempre que se desenhava um novo grafo, havia resíduos do grafo anterior que se intrometiam no novo grafo, o que levava a resultados inconsistentes. Após inúmeras tentativas de solucionar o problema, resolveu-se criar um novo processo do GraphViewer por cada grafo visualizado (matando o processo anterior antes de criar um novo), o que, apesar de não ser a mais ideal das soluções, resolveu impecavelmente o problema

Esforço de cada elemento do grupo

Tiago Lascasas dos Santos – 40%

Leonardo Gomes Capozzi – 30%

Ricardo Miguel Oliveira Rodrigues de Carvalho – 30%

Conclusão

Após a realização deste trabalho, concluímos que existem várias maneiras de abordar um mesmo problema, e que a decisão sobre qual a melhor maneira a adotar para o resolver da forma mais eficiente e apropriada pode ser tanto ou mais difícil do que a própria implementação da solução. No contexto concreto deste projeto, chegámos à conclusão que a solução apresentada para o último ponto (que é o cerne do trabalho) poderia ter sido realizada mais eficientemente se o grafo fosse não dirigido ou se tivéssemos a hipótese de usar algoritmos como os caminhos ou circuitos de Euler. A solução apresentada fornece uma boa solução para casos em que tanto o número de camiões como o número de clientes é alto, visto que quantos mais clientes houver, mais hipóteses há de um caminho passar por muitos clientes e que quantos mais camiões houver, mais eficientemente se consegue distribuir os vários caminhos entre eles, o que leva a um menor tempo de entrega.

Este projeto levou também a um melhor entendimento sobre os algoritmos em grafos estudados, nomeadamente o algoritmo de Dijkstra, visto que este está presente nas soluções de quase todos os subproblemas da premissa inicial. Foi também interessante usar pela primeira vez compilação condicional e a utilização de uma API de um programa a correr paralelamente ao nosso, como é o caso do GraphViewer. A utilização de dados reais sobre uma zona que conhecemos bem permitiu dar, também, uma melhor perspetiva e entendimento do contexto do trabalho.

Bibliografia e referências:

- Mark Allen Weiss, Data Structures and Algorithm Analysis in C++, 4th edition (Florida State University: Pearson, 2014), 386–399
- Slides disponibilizados no Moodle pelos docentes
- Fórmula de haversine: http://www.movable-type.co.uk/scripts/latlong.html
- Biblioteca ncurses: http://www.invisible-island.net/ncurses/ncurses.html