

Projecto 1 - Criação de Equipas CAL 2013/2014

Turma 06

| | | |
|---------------|-----------|------------------|
| João Neto | 201203873 | ei12013@fe.up.pt |
| João Maia | 201206047 | ei12089@fe.up.pt |
| Miguel Mendes | 201105535 | ei11058@fe.up.pt |

Índice

| | |
|------------------------------------|---|
| Criação de Equipas | 2 |
| Algoritmos Estudados | 3 |
| Algoritmo de Prim: | 3 |
| Algoritmo de Kruskal: | 4 |
| Possíveis Resoluções | 5 |
| Adaptação do Algoritmo de Prim: | 5 |
| Adaptação do Algoritmo de Kruskal: | 6 |
| Trabalho de Grupo | 7 |
| Distribuição de Tarefas | 7 |

Criação de Equipas

Descrição Informal

Tendo um grupo de especialistas G , cada um com um conjunto de competências S , e uma rede de especialistas $R(E,C)$ que representa o custo de comunicação entre cada um deles, pretende-se construir uma equipa E , pertencente a G , para executar um projecto P .

P tem um conjunto de competências necessárias, que devem ser reunidas pelos elementos de E .

O objectivo do problema é encontrar a equipa E que simultaneamente reúna estas competências e minimize o custo de comunicação entre os seus elementos.

Este custo é calculado somando os custos de comunicação da minimum spanning tree resultante do algoritmo a desenvolver.

Descrição Formal

Dados de Entrada:

G -> Grupo de Especialistas;

S -> Conjunto de Competências (Skills) de um determinado Especialista;

$R(E,C)$ -> Rede de ligação dos Especialistas, com o custo de comunicação entre si;

P -> Projecto e respectivo conjunto de Competências necessárias;

Dados de Trabalho:

C -> Conjunto de competências necessárias ao projecto que serão eliminadas à medida que forem encontrados especialistas que as reúnem;

Dados de Saída:

E -> Equipa que reúne as Capacidades necessárias e tem o custo de comunicação mínimo;

Restrições:

Competências de P contido em Competências de E ;

Objectivo:

Minimizar Somatório dos custos de Comunicação entre os elementos da equipa;

Algoritmos Estudados

Os algoritmos estudados no decorrer da Unidade Curricular para resolver o problema da árvore de expansão mínima, Prim e Kruskal, serão descritos em seguida.

Algoritmo de Prim:

Este algoritmo expande a árvore por adição sucessiva de arestas e vértices, escolhendo a aresta com menor custo que liga a um vértice que ainda não pertence à árvore. Assim, podemos classificá-lo como algoritmo ganancioso.

A informação que é necessário manter em cada vértice é a seguinte:

- Dist - Peso da aresta que liga o vértice à árvore;
- Path - Vértice que está na outra extremidade da aresta que o liga à árvore;
- Visited - Um valor booleano que indica se o vértice já pertence à árvore.

Para manter as arestas que ainda falta processar, podemos usar uma estrutura linear não ordenada, ficando o algoritmo com complexidade temporal é de $O(|V|^2)$, mas se usarmos uma fila de prioridade melhoramos a complexidade para $O(|E| \log |V|)$.

Pseudo-Código

Inicializar a fila de prioridade de vértices

Inicializar todos os vértices com a seguinte informação:

Path - Apontador nulo

Dist - Infinito

Visited - Falso

Inicializar o vértice inicial com a seguinte informação:

Dist - 0

Colocar na fila de prioridade o vértice inicial

Enquanto a fila de prioridade não estiver vazia

Retirar da fila o vértice com o menor custo de ligação, vértice u

Marcar o valor Visited de u como Verdadeiro

Percorrer todas as arestas de u

O vértice destino da aresta atual, vértice v

Se v ainda não foi visitado

Se Dist em v é infinito

Adicionar v à fila de prioridade

Se Dist em v é superior ao peso da aresta que liga u a v

Atualizar v para que:

Path - Apontador para u

Dist - Peso da aresta que liga u a v

Algoritmo de Kruskal:

Este algoritmo expande a árvore por adição sucessiva das arestas de menor peso que não criem ciclos, sendo por isso um algoritmo ganancioso.

O algoritmo mantém uma floresta de árvores, inicialmente cada vértice é uma árvore, e ao adicionar uma aresta estamos a juntar duas árvores diferentes. O algoritmo termina quando só houver uma árvore, que será a árvore de expansão mínima.

Para aceitar uma aresta precisamos de verificar se os extremos não pertencem à mesma árvore, para evitar ciclos.

A complexidade temporal deste algoritmo é equivalente à do Algoritmo de Prim com a fila de prioridade implementada, $O(|E| \log |V|)$.

Pseudo-Código

Inicializar o contador Arestas Aceites a 0, que vai manter o nº de arestas adicionadas

Adicionar as arestas a uma fila de prioridade de mínimos, ordenadas pelos seus pesos

Enquanto Arestas Aceites for menor que o nº total de arestas menos 1

 Seleccionar a aresta com o menor peso da fila de prioridades, aresta e

 Se as extremidades da aresta não pertencerem à mesma árvore

 Incrementar o contador Arestas Aceites

 Juntar as duas árvores pela aresta e

Possíveis Resoluções

Os algoritmos estudados no decorrer da Unidade Curricular, Prim e Kruskal, resolvem o problema da árvore de expansão mínima (minimum spanning tree), com todos os nós da árvore inicial. Para a resolução do nosso problema precisamos de os adaptar para que eles respeitem a restrição das competências.

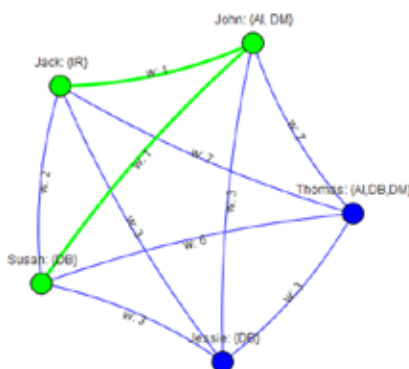
Adaptação do Algoritmo de Prim:

Para que este algoritmo respeite a restrição, podemos implementar uma lista de competências, que vai sendo alterada à medida que adicionamos especialistas à equipa, retirando as competências que esse especialista tem. Assim, a lista seria inicializada com as competências necessárias ao projeto, e a condição de paragem seria a lista estar vazia.

O algoritmo arranca a partir de um nó fornecido, o que levanta o problema da seleção desse nó. As primeiras ideias que surgiram foram as alternativas gananciosas abaixo apresentadas:

- Seleccionar nó com o maior número de competências -> Tendo inicialmente reunido o maior número de competências possíveis, o número de ligações necessárias para completar a equipa vai ser menor, potencialmente reduzindo o custo de comunicação total;
- Seleccionar nó com menor custo de comunicação total -> Tendo seleccionado o elemento que comunica melhor, aumentamos a probabilidade de escolher ligações de baixo custo, reduzindo o custo de comunicação total;

No entanto, nenhuma destas escolhas garante a resolução do problema, como podemos ver no exemplo que se segue, em que as competências necessárias ao projecto são {AI, DB, DM, IR} :



A equipa ideal é constituída pelos elementos Jack, John e Susan, com um custo de comunicação igual a 2.

O especialista com o maior número de competências necessárias ao projeto é o Thomas, e a especialista com um menor custo de comunicação total é a Jessie, mas como podemos ver, nenhum deles faz parte da equipa ideal.

Assim, concluímos que não há um método para escolher um elemento que garantidamente faça parte da equipa ideal, para a partir dele podermos correr o Algoritmo de Prim.

A solução encontrada é correr o algoritmo a partir de todos os especialistas e manter uma variável que regista o custo de comunicação da melhor equipa encontrada até agora, que é inicializada a infinito, e alterada ao fim de cada iteração do algoritmo, sendo atualizada se o custo de comunicação da equipa atual for menor do que o melhor valor encontrado nas anteriores execuções.

Adaptação do Algoritmo de Kruskal:

Para que este algoritmo resolva o problema, temos que adicionar algumas estruturas auxiliares, para manter a informação referente à evolução do mesmo:

- Lista com as competências necessárias do projecto (Lista S);
- Lista para os membros da equipa (Lista M).

Como descrito acima, o Algoritmo de Kruskal trabalha sobre uma lista de arestas ordenadas pelo seu peso. Esta lista também sofre alterações para se adaptar ao nosso problema, removendo todas as arestas que não cumprem nenhuma das seguintes características:

- Ambos os vértices da aresta contêm pessoas têm uma ou mais competências da Lista S;
- Um vértice da aresta é um membro da equipa e o outro é uma pessoa cujas competências já se encontram na Lista S.

Assim garantimos que todas as arestas da Lista acrescentam competências necessárias ao projeto.

Pseudo-Código

```
Inicializar a Lista S com as competências necessárias
Inicializar a Lista M vazia
Remover arestas redundantes
Enquanto a lista S não estiver vazia
    Retirar da Lista S a aresta com menor Peso, aresta E
    Se alguma das pessoas ligadas por E não fazem parte de M
        Adicionar as que ainda não fizeram parte da equipa a M
        Remover as competências dessa pessoa da lista S
    Remover arestas redundantes
```

Implementação

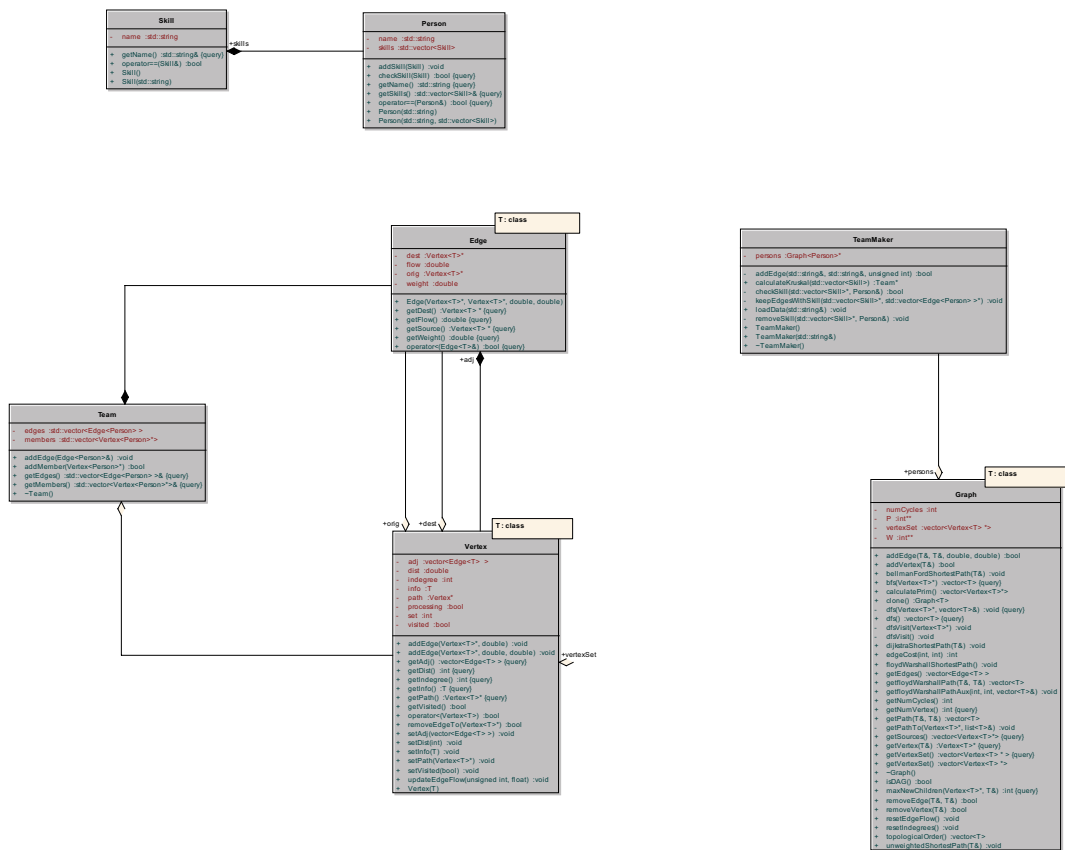
O nosso grupo implementou duas soluções para o problema, com a adaptação do Algoritmo de Prim e do Algoritmo de Kruskal. No entanto, na solução entregue o programa usa a adaptação do Algoritmo de Kruskal, por ser mais eficiente.

Casos de Utilização

Aquando da entrega, o programa não tem menus. Correndo o executável, o programa carrega os dados dos ficheiros persons1.txt, que contém os nomes dos especialistas e respetivas competências. Seguidamente resolve o problema para as competências necessárias AI, DB, DM e IR.

O resultado é mostrado através do GraphVisualizer, mostrando o algoritmo usado passo a passo.

Modelo de Classes



Trabalho de Grupo

Distribuição de Tarefas

A distribuição de tarefas foi feita tendo em conta as áreas em que cada membro se sente mais à vontade, facilitando assim todo o desenvolvimento do projeto.

Após reunião para debater o algoritmo a implementar, o grupo dividiu as tarefas da seguinte forma:

João Neto - Visualizador de Grafos e Relatório

João Maia - Implementação dos algoritmos

Miguel Mendes - Input a partir de ficheiros

Dificuldades Encontradas

A implementação dos algoritmos foi dificultada pela ineficácia de debugging do IDE utilizado, pelo que se recorreu a outro IDE para fazer esta tarefa.