Relatório do 2º projeto de ASA

1. Introdução

O problema proposto pode ser representado através de um grafo não dirigido e pesado, G_A.

Em G_A, cada cidade é representada por um vértice. Cada potencial estrada é uma aresta entre dois vértices representativos de cidades, com peso igual ao custo de a construir. Existe ainda um vértice adicional, V_{Aero}, tal que uma aresta entre uma cidade e V_{Aero} representa a potencial construção de um aeroporto na cidade, e tem peso igual ao seu custo de construção.

O objetivo é encontrar um conjunto de estradas e/ou aeroportos que permita ligar todas as cidades, com o menor custo possível. Se a solução tiver pelo menos um aeroporto, então é uma árvore de menor custo do grafo descrito. Se a solução não tiver nenhum aeroporto, então é uma árvore de menor custo de um grafo G_E, que é igual ao grafo G_A eliminando o vértice V_{Aero} (e as arestas a ele adjacentes).

2. Descrição da solução

2.1 Ordenação das arestas

Durante a execução do algoritmo de Kruskal, é feita uma ordenação das arestas do grafo do qual se quer obter uma MST. Como tal, as arestas interpretadas do input são guardadas em dois *arrays*, sob a forma de uma estrutura. Um dos *arrays*, E_E, contém as arestas que correspondem a estradas entre duas cidades, e o outro, E_A, contém as arestas entre uma cidade e o vértice V_{Aero}, que correspondem aos aeroportos. Assim, tem-se que o conjunto das arestas de G_E é E_E, e o conjunto das arestas de G_A é o resultado da concatenação de E_E com E_A.

Para evitar que as arestas em E_E sejam ordenadas duas vezes, estes *arrays* são ordenados separadamente, sendo depois concatenados, aquando da execução do algoritmo de Kruskal sobre G_A, usando uma operação *merge*, estável, como a do *mergesort*. A estabilidade da operação é relevante, pois implica que a comparação de arestas é feita dando prioridade às arestas que representam estradas, i. e., se duas arestas de E_E e E_A têm pesos iguais, no *array* resultante, a aresta de E_E aparecerá antes da de E_A. Como tal, durante a execução do algoritmo de Kruskal, é priorizada a construção de estradas, em detrimento dos aeroportos.

2.2 Obtenção das MSTs

Para obter a MST de cada um dos grafos, é usado o algoritmo de Kruskal, utilizando estruturas de representação de conjuntos disjuntos que obedecem às heurísticas de compressão de caminhos e união por categorias. A execução do algoritmo retorna três valores inteiros para cada grafo: o custo total da MST gerada a partir dele, e os números de estradas e de aeroportos construídos na rede que representa.

2.3 Validação das soluções

A solução obtida deve ser tal que todas as cidades fiquem ligadas em rede. No domínio especificado, isto corresponde a ter um grafo ligado que a represente, i. e., que todos os vértices sejam atingíveis a partir de qualquer outro.

Numa árvore, o número de arestas E é sempre E = V−1, onde V é o número de vértices da mesma árvore. Se, no grafo gerado pelo algoritmo de Kruskal, o número de arestas for V − 1, então garantimos que o grafo original era ligado.

Então, durante a obtenção das MSTs de G_A e G_E, vão sendo contadas as arestas seguras identificadas pelo algoritmo de Kruskal. No final, verificamos se E = V − 1. Se não for, o input é considerado insuficiente (pois significa que o grafo original não era ligado).

2.4 Seleção da melhor solução

Caso apenas uma das soluções tenha sido validada, essa é a escolhida. Se, pelo contrário, ambas forem consideradas válidas, a escolha é feita baseada nos valores de custo total retornados. Se estes diferirem, é escolhida a solução de menor custo. Caso contrário, é selecionada a solução que minimiza o número de aeroportos a construir, que é, neste caso, a que corresponde à MST obtida a partir do grafo G_E, visto que esta não visa a construção de qualquer aeroporto. De notar que, se o número de aeroportos do input for inferior a 2, não é executado o algoritmo de Kruskal sobre G_A, uma vez que um único aeroporto não liga nenhum par de cidades. Os valores de custo total, número de estradas a construir e número de aeroportos a construir apresentados, são os obtidos da MST escolhida.

3. Análise teórica

3.1 Correção da solução

Pretende-se obter um subconjunto das arestas de um grafo ligado, pesado e não-dirigido, tal que todos os vértices sejam atingíveis de qualquer outro, não haja ciclos e cuja soma dos custos das arestas seja mínima. Isto corresponde a uma MST, que pode ser obtida pelo algoritmo de Kruskal.

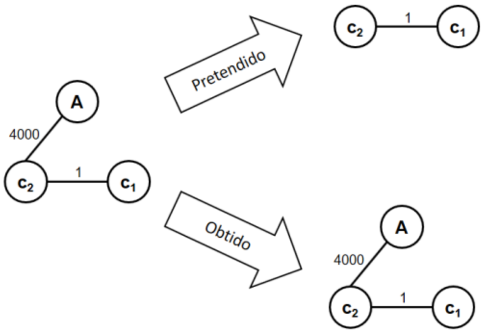

Fig. 1

A execução do algoritmo de Kruskal sobre o grafo G_A apenas determina a solução correta se a solução implicar a construção de aeroportos. Caso contrário, o grafo obtido terá necessariamente pelo menos um aeroporto (se o input permitir a sua construção) ou não será ligado, o que não é o pretendido (Fig. 1). Existem também situações em que a solução pretendida não tem aeroportos, mas o algoritmo de Kruskal devolve um grafo que sugere a sua construção, que acontece quando há soluções com e sem aeroportos com o mesmo custo (Fig. 2).

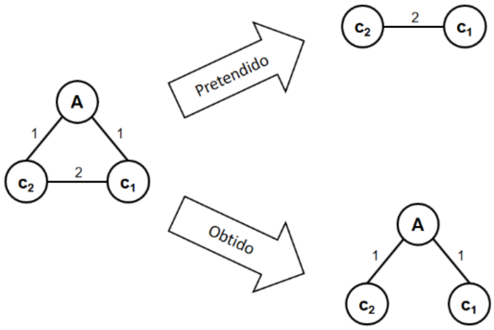

Fig. 2

Assim, não existe nenhum critério que permita decidir se o grafo obtido é o pretendido antes de compararmos os resultados das duas execuções dos algoritmos. É possível, contudo, garantir a correção da solução quando implica a existência de aeroportos – o algoritmo de Kruskal garante que se obtém uma árvore de menor custo.

Caso a solução contemple apenas a construção de estradas, não é preciso um grafo que represente os aeroportos, e executa-se o algoritmo de Kruskal sobre G_E. O resultado do algoritmo é o correto se a solução apenas contemplar a construção de estradas.

3.2 Complexidade do algoritmo

Seja:
N – O número de cidades do Bananadistão.
A – Número de potenciais aeroportos a construir.
E – Número de potenciais estradas a construir.

Então, a complexidade do algoritmo utilizado é:

1. Inicialização das listas de arestas que representam G_A e G_E O(A+E)
2. Aplicação do algoritmo de Kruskal[1] sobre G_E O(E log N)
3. Aplicação do algoritmo de Kruskal[2] sobre G_A O(A log N)

Conclusão: o algoritmo tem complexidade O((A+E) log N).

4. Avaliação experimental dos dados

4.1 Teste 1

O primeiro teste executado teve o objetivo de estudar a variação do tempo de execução do algoritmo (em *clocks* do processador) em função do número de vértices (N) dos grafos criados a partir do input.

O único passo do algoritmo que depende do número de vértices do grafo é o conjunto das operações de *make-set* executadas na inicialização do algoritmo de Kruskal. São executadas N operações de *make-set*, cada uma em tempo constante. Como tal, é esperado que o tempo de execução varie linearmente em função do número de vértices dos grafos gerados a partir do input.

O input fornecido contemplava apenas a construção de estradas, com o intuito de permitir a obtenção de valores mais precisos (a relação entre os números de aeroportos e estradas não teve de ser tida em consideração). O programa foi executado 56 vezes com os seguintes valores de input:

- E = 160 000;
- A = 0;
- $\sqrt{160\,000} < N < 160\,000$;
- Custo máximo = 160 000;

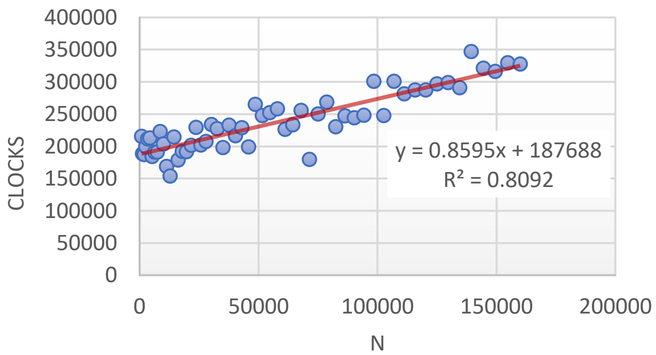

Gráfico 1 - Tempo em função do número de vértices, N

[1] Implementado com árvores com compressão de caminhos e união por categorias.

[2] Implementado com árvores com compressão de caminhos e união por categorias, e reutilizando os arcos já ordenados de G_E (ver 2.1)

4.2 Teste 2

O segundo teste foi executado com o intuito de estudar o comportamento do algoritmo com a variação do número de arestas dos grafos gerados a partir do input.

Na execução do algoritmo de Kruskal sobre G_E, é realizada uma ordenação das arestas, com complexidade O(E logE), e operações de *find-set* e *union*, com complexidade O(E). Na execução do algoritmo de Kruskal, é executada uma ordenação do *array* de arestas-aeroporto, com complexidade O(A log A), seguida da operação de *merge* dos dois arrays (ver 2.12.1), de complexidade O(A + E). Por fim, são executadas O(A) operações de *find-set* e *union*.

Pode concluir-se que a complexidade temporal do algoritmo em função de número de arestas, A + E, é O((A+E) log(A+E))[3].

Para testar este resultado, o programa foi executado 100 vezes com os seguintes valores de input:

- 40 < N < 4000, variando linearmente;
- 0 < A < N, calculado aleatoriamente;
- E = $\frac{N^2}{2} - A$;
- Custo máximo da rede = $\frac{N^2}{2}$;

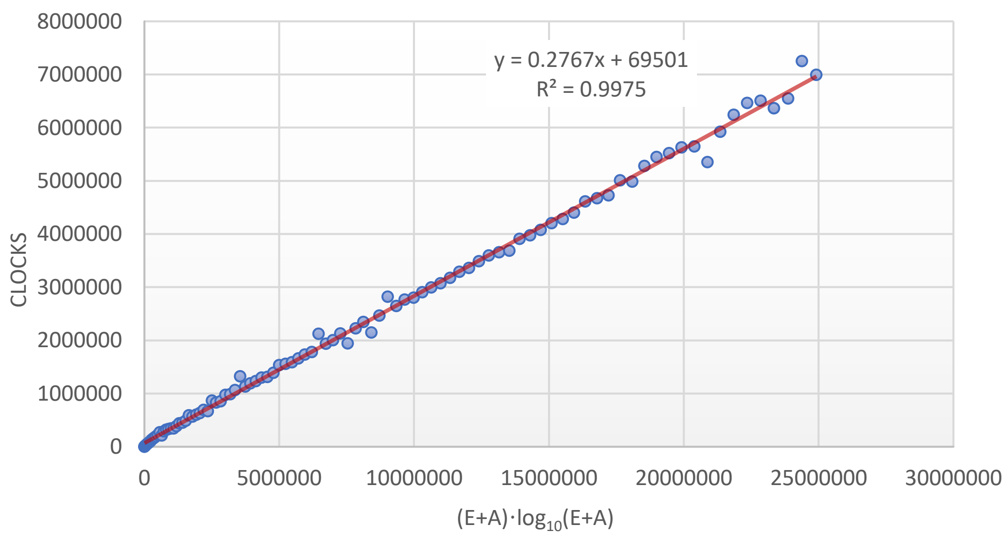

Gráfico 2 - Tempo de execução em função do número de arestas, E+A

4.3 Conclusão

Dos testes executados, conclui-se que o algoritmo é O(N + (A + E) log(A + E)). Como, nos casos em que o input não é insuficiente, o grafo é ligado, tem-se que A + E ≥ N – 1, de onde se tira que o algoritmo é também O((A + E) log(A + E)). Dado que A + E < N^2, tem-se também que o algoritmo é O((A+E) log(N)), como previsto em 3.2.

---

[3] O(E log E) + O(E) + O(A log A) + O(A + E) + O(A) ⇒ O(E log E + A log A) ⇒
⇒ O(E log (A + E) + A log (A + E)) ⇒ O((A+E) log(A+E))