TP1 - Grafos planares

17 pontos

Entrega: 17/09/2023

1 Objetivo do trabalho

Neste trabalho, vamos exercitar tópico relativos ao primeiro terço do curso: grafos e alguns algoritmos elementares em nessas estruturas. Para tal, trabalharemos com um problema em uma classe de grafos conhecida como *grafos planares*.

Serão fornecidos alguns casos de teste para que você possa testar seu programa, mas eles não são exaustivos! Podem haver situações que não são ilustradas por eles; cabe a você pensar em novos casos e garantir que seu programa esteja correto e implemente um algoritmo de complexidade adequada.

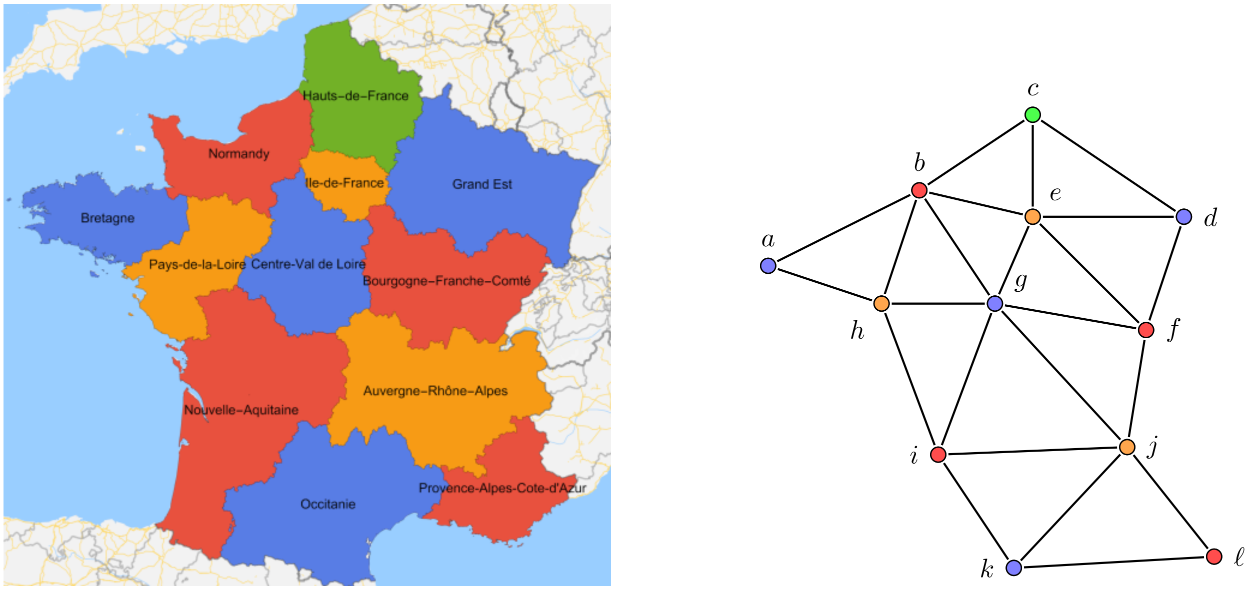
1.1 Informações importantes

O código fonte do seu trabalho deve estar contido em um **único** arquivo na linguagem C++ e deve ser submetido via Moodle na tarefa `Entrega TP1` até o dia 17/09/2023. Você terá 10 tentativas para conseguir a nota total de execução; apenas a última submissão será levada em conta para fins de avaliação. Você não terá acesso a todos os casos de teste; determinar estratégias para testar seu programa e suas ideias faz parte do trabalho. Envios com atraso serão aceitos; leia a Seção 5 para a política de atrasos.

Plágio de qualquer natureza não será tolerado. Caso qualquer cola seja encontrada, seu trabalho será zerado e as demais providências cabíveis serão tomadas. Escreva seu próprio código, de maneira legível e com comentários apropriados; ele pode ser útil no futuro próximo.

2 Definição do problema

No século XIX, era muito comum que as pessoas inventassem coisas para fazer durante o tempo livre, já que elas não tinham internet nem nada do tipo, e os meios de transporte mais rápidos do dia-a-dia eram movidos a tração animal (bois, cavalos, etc). Um passa-tempo aparentemente bem comum era a coloração de mapas: afinal de contas, é muito mais fácil conseguir identificar fronteiras ou outras áreas de interesse se regiões adjacentes estiverem coloridas com cores diferentes. Veja, por exemplo, o mapa da França na Figura 1:



(a) (b)

Figure 1: (a) Mapa da França onde cada região tem uma cor diferente de todas as suas vizinhas; (b) grafo que representa o mapa à esquerda com as cores apropriadas.

Depois de muitas pessoas diferentes brincarem de colorir mapas, elas repararam que nunca precisaram de mais de 4 cores e durante mais de um século não sabíamos se existia um mapa que precisa de uma quinta cor. Esse problema só foi resolvido no fim do século XX e hoje é conhecido como teorema das quatro cores.

Este trabalho não tem (quase) nenhuma relação com o teorema em si, mas esse último motivou o estudo de grafos muito importantes hoje em dia: os grafos planares. Um grafo é dito *planar* se existe um jeito de desenhar seus vértices e arestas numa folha de papel sem que arestas se cruzem. Esses grafos tem propriedades fundamentais para a construção de algoritmos eficientes; a mais interessante de todas para nós neste momento é a existência do que chamamos de *faces*: regiões delimitadas por arestas que não são inteiramente subdividas por nenhuma outra aresta do grafo. Podemos enxergar as faces como regiões diferentes do plano e os vértices/arestas que delimitam a face estão na região chamada *borda* da face, que é o tema principal deste trabalho. Por exemplo, o grafo da Figura 2 possui cinco faces.

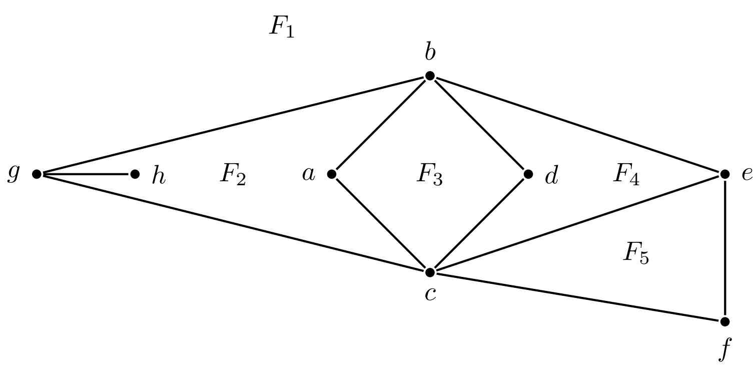


Figure 2: Um grafo e suas cinco faces F_1, F_2, F_3, F_4, F_5.

Note que F_1 é composta sempre pelos vértices de fora do grafo e por isso é chamada de face *externa* do grafo. Com isso, temos que cada face é uma sequência de vértices onde o primeiro e o último são iguais e dois vértices consecutivos são adjacentes. Ou seja, podemos descrever as faces do grafo acima como: $F_1 = \langle b, e, f, c, g, b \rangle$, $F_2 = \langle c, g, h, g, b, a, c \rangle$, $F_3 = \langle a, b, d, c, a \rangle$, $F_4 = \langle e, c, d, b, e \rangle$, e $F_5 = \langle f, c, e, f \rangle$. Repare também que um grafo planar pode ser desenhado de diferentes maneiras no plano e a forma de desenho **interfere nos vértices que compõem cada face do grafo**; por exemplo, no grafo da Figura 2, podemos mover h para a esquerda de G, fazendo com que h participe da face externa mas deixe de participar da face F_2.

Neste trabalho, dado um grafo planar G e as coordenadas de seus vértices em um desenho válido de G (ou seja, sem cruzamento de arestas e todas as arestas sendo segmentos de reta entre os pontos de seus extremos), você deve listar **todas as faces de** G.

3 Dicas

Existem várias soluções possíveis para esse problema, algumas bem complicadas e outras (esperadas) mais simples. Todas elas exigem um pouquinho de geometria computacional, por isso abaixo tem algumas funções que podem ser úteis.

```
struct Ponto {
  double x, y;
};

// Distância euclidiana de a para b.
double Distancia(Ponto a, Ponto b) {
  double x = (a.x - b.x), y = (a.y - b.y);
  return sqrt(x*x + y*y);
}

// Coeficiente da reta que passa na origem e p.
double Inclinação(Ponto p) {
  return atan2(p.y, p.x);
}

// Coeficiente da reta orientada de p para q.
double InclinaçãoRelativa(Ponto p, Ponto q) {
  return atan2(q.y - p.y, q.x - p.x);
}

// Determina se ao caminhar de a para b e depois de b para c
estamos fazendo uma curva à esquerda, à direita, ou seguindo em frente.
int TipoCurva(Ponto a, Ponto b, Ponto c) {
    double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
    if (v < 0) return -1; // esquerda.
    if (v > 0) return +1; // direita.
    return 0; // em frente.
}
```

Abaixo vão algumas perguntas relevantes para você se fazer antes de começar a programar:

Pergunta 1. *A quantas faces uma dada aresta pode pertencer?*

Pergunta 2. *Dado que acabei de atravessar uma aresta e de u para v, existe alguma outra aresta incidente v que com certeza está em uma mesma face que e?*

Pergunta 3. *Como posso evitar de passar por uma mesma face mais de uma vez?*

Para responder a essas perguntas, olhe novamente para a Figura 1 e tente pensar em como podemos gerar todas as faces que contém o vértice a se a primeira aresta que pegarmos na hipótese da questão 2 for incidente a a.

Pergunta 4. *Será que se fizer alguma transformação nas arestas minha vida fica mais fácil?*

Dica da dica: pense em orientações/curvas para essa última pergunta.

4 Casos de teste

4.1 Formatado da Entrada

Cada caso de teste é composto por várias linhas. A primeira linha contém dois inteiros, N e M, que representam, respectivamente, o número de vértices e arestas do grafo de entrada G. É garantido que G é conexo, que $1 \leq N, M \leq 10^5$, e que $V(G) = \{1, 2, \ldots, N\}$. A i-ésima linha da entrada começa com dois números reais x_i, y_i, que representam as coordenadas do vértice i no plano cartesiano; é garantido que $-10^4 \leq x_i, y_i \leq 10^4$ Na mesma linha, temos um inteiro positivo d_i, que representa o grau do vértice i. Ainda na mesma linha, temos d_i inteiros entre 1 e N, cada um correspondendo a um vizinho de i; é garantido que um vértice não é vizinho de si mesmo.

4.2 Formato da Saída

A primeira linha da saída contém um inteiro F, que deve ser igual ao número de faces de G. As próximas F linhas devem corresponder às F faces de G; cada linha começa com um inteiro s_i, que representa o tamanho da i-ésima face de G; em seguida, existem s_i inteiros, representando o circuito que corresponde à borda da face. A ordem das faces não é importante, mas vértices consecutivos na borda da face devem ser adjacentes.

4.3 Limites de execução

Para qualquer caso de teste, seu código deve imprimir a resposta em no máximo 3 segundos. Seu programa deve usar menos de 100MB de memória. Estruturas de dados devem ser alocadas sob demanda; ou seja, não faça vetores estáticos gigantescos para grafos com poucos vértices. Todas as avaliações serão feitas automaticamente via VPL. Programas que não aderirem a essas restrições para um teste terão a nota do mesmo zerada.

Lembre-se: você pode submeter uma solução para a tarefa no máximo 10 vezes e apenas a última submissão será levada em conta para fins de avaliação.

4.4 Exemplos

4.4.1 Exemplo da Figura 2

Na figura abaixo, o vértice de número i corresponde à i-ésima letra do alfabeto;, i.e. $4 = d$.

Entrada	Saída
8 11	5
0 0 2 2 3	6 2 5 6 3 7 2
1 1 4 1 4 5 7	7 3 7 8 7 2 1 3
1 -1 5 1 4 5 6 7	5 1 2 4 3 1
2 0 2 2 3	5 5 3 4 2 5
4 0 3 2 3 6	4 6 3 5 6
4 -1.5 2 3 5	
-3 0 3 2 3 8	
-2 0 1 7	

4.4.2 Exemplo da Figura 1b

Na figura abaixo, o vértice de número i corresponde à i-ésima letra do alfabeto;, i.e. $4 = d$.

Entrada
12 23
-3 1 2 2 8
-1 2 5 1 3 5 7 8
0.5 3 3 2 5 4
2.5 1.65 3 3 5 6
0.5 1.65 5 2 3 4 6 7
2 0.15 4 4 5 7 10
0 0.5 6 2 5 6 8 9 10
-1.5 0.5 4 1 2 7 9
-0.75 -1.5 4 7 8 10 11
1.75 -1.4 5 6 7 9 11 12
0.25 -3 3 9 10 12
2.9 -2.85 2 10 11

Saída
13
11 1 8 9 11 12 10 6 4 3 2 1
4 1 2 8 1
4 3 2 5 3
4 2 8 7 2
4 2 5 7 2
4 3 4 5 3
4 5 7 6 5
4 5 4 6 5
4 7 8 9 7
4 7 9 10 7
4 7 6 10 7
4 9 10 11 9
4 10 11 12 10

4.4.3 Exemplo Bônus

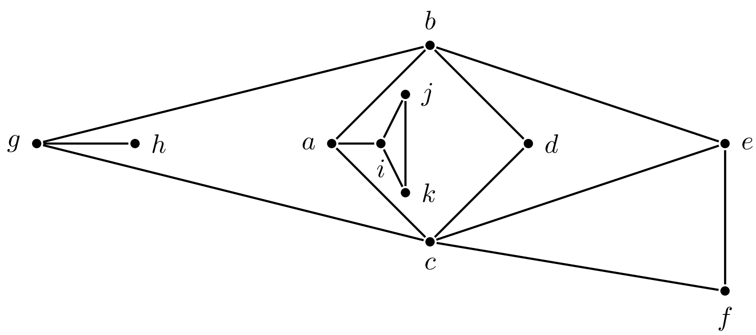


Figure 3: Figura do exemplo bônus.

Entrada
11 15
0 0 3 2 3 9
1 1 4 1 4 5 7
1 -1 5 1 4 5 6 7
2 0 2 2 3
4 0 3 2 3 6
4 -1.5 2 3 5
-3 0 3 2 3 8
-2 0 1 7
0.5 0 3 1 10 11
0.75 0.5 2 11 9
0.75 -0.5 2 10 9

Saída
6
10 1 3 4 2 1 9 10 11 9 1
7 1 2 7 8 7 3 1
6 2 5 6 3 7 2
5 2 4 3 5 2
4 3 6 5 3
4 9 11 10 9

5 Atrasos

O trabalho pode ser entregue com atraso, mas será penalizado de acordo com a seguinte fórmula, onde d é o número de dias atrasados:

$$\Delta(d) = \frac{2^{d-1}}{0.32}\% \tag{1}$$

Por exemplo, com um atraso de quatro dias e uma nota de execução de 70% do total, sua nota final será penalizada em 25%, ficando assim igual a $70 \cdot (1 - \Delta(d)) = 52.5\%$. Note que a penalização é exponencial, e um atraso de 6 dias equivale a uma penalidade de 100%.

6 Errata

- Correção na função `Inclinação`.
- Adição da função `InclinaçãoRelativa`.
- Correção exemplo 2 e adição de novo exemplo.
- Correção saída exemplo bônus e descrição da saída.