TP2 - Os Reis Amarelos

17 pontos

Entrega: 28/04/2024

1 Objetivo do trabalho

Neste trabalho, vamos exercitar tópico relativos ao segundo terço do curso: grafos e alguns algoritmos clássicos nessas estruturas.

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 que ele 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 TP2` até o dia 28/04/2024. Você terá 30 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 6 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

A Bacônia é, hoje, uma sociedade vibrante e culturalmente muito desenvolvida. Um dos preceitos principais do Deboismo, a filosofia dominante entre as capivaras que compõem a maior parte dos habitantes da Bacônia, é maximizar o tempo que você fica de boa: aproveitando a vida, descansando, e até mesmo aprendendo coisas novas puramente pelo prazer e fazê-lo.

Para tornar o processo de aprendizado da história do reino ainda mais interessante, as capivaras da Divisão de Conhecimento da Coletividade fizeram o que fazem de melhor: desenvolveram um complexo jogo de cartas, que resolveram chamar de *Os Reis Amarelos*, ou ORA, para os fans. Uma campanha de ORA é divida em vários cenários, que exploram os mais diferentes episódios da história da Bacônia, como a gloriosa reunificação alcançada por Bacon - O Grande, ou os mistérios combinatórios com ramificações sobrenaturais encontrados por Bacon - O Grafo.

Cada cenário é jogado em cima de um mapa, e tem diferentes objetivos intermediários, como encontrar a tumba perdida da dinastia Bacon ou enfrentar os mais temidos inimigos das capivaras: os jacarés do papo-amarelo (que dão o nome ao jogo), carinhosamente apelidados de *monstros*. Independente destes objetivos, ao fim de cada cenário os jogadores passam pela **fase de fuga**, onde eles devem:

- Fugir para a segurança de Bacopolis, a capital do reino;
- Durante a fuga, eles não podem dividir um espaço no mapa com um inimigo;
- Caso a fuga não seja bem sucedida, os jogadores devem maximizar a quantidade de turnos jogados.
- A quantidade de recursos gastos durante a fase de fuga deve ser minimizada.

- A fuga deve ser completada em, no máximo, T turnos.

O jogo é muito muito complexo, e felizmente ficou a nosso cargo implementar apenas um verificador de vitoria para a fase de fuga! Vamos ver como funcionam as diferentes mecânicas do jogo nessa fase:

Os turnos. O primeiro turno jogado é o turno 1. Jogam, alternadamente, os jogadores e, imediatamente em seguida, os monstros. No início do turno dos jogadores, eles recebem K recursos extras ao todo. Se o jogo chegar no turno $T + 1$ e os jogadores não estiverem em Bacopolis, eles perdem o jogo. Se os jogadores chegarem a Bacopolis, o jogo termina **imediatamente**.

O mapa. Cada espaço do mapa possui uma lista de destinos possíveis, que **não necessariamente é simétrica**: tematicamente, os jogadores podem saltar do alto da torre do castelo de Bacon - O Mal - para o rio que passa ao lado do castelo, mas não podem ir diretamente do rio para o alto da torre. Há também um custo para se ir do lugar u para o lugar v: caso os jogares não possuam recursos suficientes para ir de u para v, eles não podem tomar esse caminho. Bacopolis está localizada no espaço de número N.

Os jogadores. Os jogadores, coletivamente, controlam uma única caravana, que tem como ponto inicial o espaço de número 1 do mapa. A cada turno, os jogadores escolhem um espaço adjacente que não esteja ocupado por monstros **naquele momento** e se movem para lá; **os jogadores podem ficar parados a um custo de** 1 **recurso por turno**.

Os monstros. Cada um dos J monstros tem um espaço inicial, garantidamente diferente do espaço inicial da caravana. Antes do primeiro turno, cada monstro traça uma rota de comprimento mínimo de sua posição para **o espaço original da caravana**; monstros tem uma predileção por caminhos lexicograficamente pequenos, ou seja, se existem os caminhos $5 \to 6 \to 7$, e $5 \to 8 \to 7$, um monstro irá tomar o primeiro caminho, e não o segundo, pois ambos tem o mesmo comprimento, mesma origem e destino, mas o primeiro usa o espaço 6 ao invés do 8. Na sua vez, o monstro j avança uma posição na sua rota pré-estabelecida; se ele parar na mesma posição da caravana, o jogo termina **imediatamente** e os jogadores são derrotados; se ele chegar no seu destino final, ele fica parado e não se move mais. Uma quantidade arbitrária de monstros podem ocupar um mesmo espaço. Monstros ignoram os custos de ir de um espaço para outro e jogam em ordem: primeiro o monstro de número 1, depois o de número 2, e assim por diante. Se um monstro move-se para o espaço onde os jogadores estão atualmente, o jogo termina **imediatamente**.

A DCC gostaria de saber, dada uma configuração do início da fase de fuga, o que os monstros vão fazer e o que vai acontecer se os jogadores agirem de forma ótima. Veja a Figura 1 e sua descrição para um exemplo.

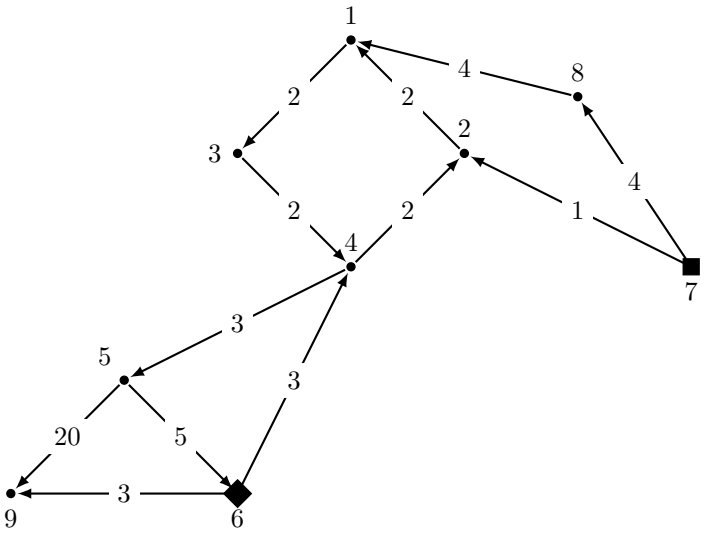

Figure 1: Uma possível configuração de inicial de um mapa do jogo *Os Reis Amarelos* onde todas . Inicialmente, temos dois monstros: um no espaço 6 e outro no espaço 7. O monstro no espaço 6 traçará o caminho $6 \to 4 \to 2 \to 1$, enquanto o do monstro do espaço 7 traçará $7 \to 2 \to 1$. Os jogadores tem como rota ótima os espaços $1 - 1 \to 3 \to 4 \to 5 \to 6 \to 9$, com um custo total de $1 + 2 + 2 + 3 + 5 + 3 = 16$ unidades. Note que, se o limite de turnos for $T = 5$, os jogadores devem seguir o caminho $1 - 1 \to 3 \to 4 \to 5 \to 9$, mas para isso precisam de passar pela conexão $5, 9$, a custo 20, o que pode ser impossível caso a quantidade de recursos por turno seja pequena. Por exemplo, caso os jogadores recebam apenas 2 recursos por turno e tenham 5 turnos para fugir, eles farão o caminho $1 - 1 \to 3 - 3 - 3 - 3$, que é o único caminho que maximiza a quantidade de turnos jogados e minimiza os recursos gastos.

3 Dicas

Um segundo equivale a, aproximadamente, 10^8 operações em um computador moderno. Ou seja, um algoritmo $\mathcal{O}(N)$ para $N \leq 10^6$ cabe em um segundo, mas um algoritmo $\mathcal{O}(N^2)$ não cabe.

Operações de entrada e saída são extremamente caras: use as funções `scanf` e `printf` ao invés das funções `cin, cout`.

Tempo e espaço são apenas dimensões do universo.

4 Casos de teste

4.1 Formatado da Entrada

Cada caso de teste é composto por várias linhas, que representam um mapa de ORA. A primeira linha contém quatro inteiros, N, M, J, T, e K, que representam, respectivamente, o número de espaços no mapa, as conexões entre eles, o número de monstros, o número máximo de turnos a serem jogados, e o número de recursos dados aos jogadores por turno. É garantido que $1 \leq N \leq 10^4$, $1 \leq M \leq \min\{N^2, 10^5\}$, que os espaços são rotulados com todos os elementos do conjunto $\{1, 2, \ldots, N\}$, que $1 \leq J \leq 100$, que $1 \leq T \leq 10^4$, que $T * N \leq 10^6$, e que $1 \leq K \leq 1000$.

A segunda linha da entrada contém J inteiros, $j_1, \ldots, j_J$, com j_i representando a posição inicial do i-ésimo monstro no mapa.

Seguem-se então M linhas; a i-ésima dessas linhas contém três inteiros x_i, y_i, c_i, que representam que o espaço x_i pode nos levar ao espaço y_i com um custo de c_i recursos. É garantido que $x_i, y_i \in \{1, \ldots, N\}$ e que $1 \leq c_i \leq 1000$.

4.2 Formato da Saída

Preste muita atenção a essa seção.

4.2.1 Dando o resultado do jogo

A primeira linha da saída contém um único inteiro V, que deve ser 1 caso seja possível os jogadores ganharem o jogo e 0 caso contrário.

4.2.2 Listando os monstros

As próximas J linhas descrevem as rotas que os monstros tomam ao longo do jogo. A i-ésima dessas linhas contém $\ell_i + 1$ inteiros; o primeiro inteiro deve ser ℓ_i, que indica o comprimento da rota do i-ésimo monstro. Os próximos ℓ_i inteiros dessa linha são a rota em si, que deve ser apresentada **na ordem que o monstro a percorreu**. Lembre-se que essa rota deve ser **lexicograficamente mínima entre todas as rotas de comprimento** ℓ_i. As rotas devem ser impressas **independentemente de seu comprimento**, mesmo que qualquer rota viável para o destino do monstro seja maior que o número de turnos disponíveis no jogo.

4.2.3 O caminho dos jogadores

A próxima linha da saída contém dois inteiros R e F; o inteiro R indica o número de recursos gastos pelos jogadores ao longo do jogo, e F o número de turnos jogados. A próxima linha contém $F + 1$ inteiros. O primeiro desses inteiros deve corresponder à posição inicial da caravana. O i-ésimo dos demais inteiros corresponde ao destino dos jogadores durante o i-ésimo turno. Para facilitar a sua vida, a DCC deixou que vocês reportem **qualquer caminho que otimize os objetivos desejados**. Note que, se houver saída, queremos apenas minimizar o custo do caminho viável. Se não houver como fugir, primeiro devemos maximizar o número de turnos jogados e, dentre todos os caminhos que satisfazem a primeira condição, queremos um que minimize o custo de recursos.

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 30 vezes e apenas a última submissão será levada em conta para fins de avaliação.

4.4 Exemplos

4.4.1 Exemplo da Figura 1

Entrada
```
9 12 2 6 4
6 7
1 3 2
2 1 2
3 4 2
4 2 2
4 5 3
5 6 5
5 9 20
6 4 3
6 9 3
7 2 1
7 8 4
8 1 4
```

Saída
```
1
4 6 4 2 1
3 7 2 1
16 6
1 1 3 4 5 6 9
```

4.4.2 Exemplo 2

Neste exemplo, temos que os recursos são insuficientes para que os jogadores cheguem a tempo em Bacopolis, e por isso nem tentarão chegar à segurança da capital.

Entrada
```
9 12 2 6 2
6 7
1 3 2
2 1 2
3 4 2
4 2 2
4 5 3
5 6 5
5 9 20
6 4 3
6 9 3
7 2 1
7 8 4
8 1 4
```

Saída
```
0
4 6 4 2 1
3 7 2 1
6 5
1 1 3 3 3 3
```

5 Avaliação

Se sua saída responder corretamente os itens da seção 4.2.2, seu caso de teste receberá 30% da pontuação. Se, **além disso**, ela responder aos itens das seções 4.2.1 e 4.2.3 corretamente, ela receberá os demais 70%.

6 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%.

7 Errata

- Typo na avaliação.
- Typo saída exemplo 1.
- Correção saída exemplo 2.