Caminhos mínimos

Algoritmo de Bellman-Ford: problemas resolvidos

Prof. Edson Alves - UnB/FGA

2019

Sumário

UVA 10959 – The Party, Part I

Problema

Don Giovanni likes to dance–especially with girls! And everyone else in the party enjoyed the dance, too. Getting a chance to dance with the host (that is Don Giovanni) is the greatest honour; failing that, dancing with someone who has danced with the host or will dance with the host is the second greatest honour. This can go further. Define the Giovanni number of a person as follows, at the time after the party is over and therefore who has danced with whom is completely known and fixed:

1. No one has a negative Giovanni number.
2. The Giovanni number of Don Giovanni is 0.
3. If a person p is not Don Giovanni himself, and has danced with someone with Giovanni number n, and has not danced with anyone with a Giovanni number smaller than n, then p has Giovanni number $n + 1$

Problema

4. If a person's Giovanni number cannot be determined from the above rules (he/she has not danced with anyone with a finite Giovanni number), his/her Giovanni number is ∞. Fortunately, you will not need this rule in this problem.

Your job is to write a program to compute Giovanni numbers.

Entrada e saída

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

The first line has two numbers P and D; this means there are P persons in the party (including Don Giovanni) and D dancing couples ($P \leq 1000$ and $D \leq P(P-1)/2$). Then D lines follow, each containing two distinct persons, meaning the two persons has danced. Persons are represented by numbers between 0 and $P-1$; Don Giovanni is represented by 0.

As noted, we design the input so that you will not need the ∞ rule in computing Giovanni numbers.

Entrada e saída

We have made our best effort to eliminate duplications in listing the dancing couples, e.g., if there is a line "4 7" among the D lines, then this is the only occurrence of "4 7", and there is no occurrence of "7 4". But just in case you see a duplication, you can ignore it (the duplication, not the first occurrence).

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

Output $P-1$ lines. Line i is the Giovanni number of person i, for $1 \leq i \leq P-1$. Note that it is $P-1$ because we skip Don Giovanni in the output.

Exemplo de entradas e saídas

Sample Input

```
1

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

Sample Output

```
1
1
2
2
```

Solução com complexidade $O(T(V + E))$

- Observe que o número de Giovanni é, de fato, a distância do nó em questão até o nó zero, em número de arestas
- Esta distância pode ser computada por meio de uma BFS
- Como há garantia de conectividade do grafo de entrada (condição 4 do problema), uma única travessia é suficiente
- Como a complexidade da BFS é $O(V + E)$, a complexidade da solução é $O(T(V + E))$, onde T é o número de casos de teste

Solução com complexidade $O(T(V+E))$

```cpp
#include <bits/stdc++.h>

using namespace std;

const int MAX { 1010 };

vector<int> adj[MAX];
bitset<MAX> found;

vector<int> solve(int N)
{
    vector<int> ans(N);
    queue<int> q;

    q.push(0);
    found[0] = true;
    ans[0] = 0;

    while (not q.empty())
    {
        auto u = q.front();
```

Solução com complexidade $O(T(V + E))$

```
22          q.pop();
23
24          for (const auto& v : adj[u])
25          {
26              if (not found[v])
27              {
28                  found[v] = true;
29                  ans[v] = ans[u] + 1;
30                  q.push(v);
31              }
32          }
33      }
34
35      return ans;
36  }
37
38  int main()
39  {
40      ios::sync_with_stdio(false);
41
42      int T;
```

Solução com complexidade $O(T(V+E))$

```
43      cin >> T;
44
45      for (int test = 0; test < T; ++test)
46      {
47          found.reset();
48
49          for (int i = 0; i < MAX; ++i)
50              adj[i].clear();
51
52          int P, D;
53          cin >> P >> D;
54
55          while (D--)
56          {
57              int x, y;
58              cin >> x >> y;
59
60              adj[x].push_back(y);
61              adj[y].push_back(x);
62          }
63
```

Solução com complexidade $O(T(V+E))$

```
64          auto ans = solve(P);
65
66          if (test)
67              cout << '\n';
68
69          for (int i = 1; i < P; ++i)
70              cout << ans[i] << '\n';
71      }
72
73      return 0;
74  }
```

URI 1205 – Cerco a Leningrado

Problema

A cidade de São Petersburgo mudou de nome depois da revolção russa em 1914 para Petrogrado. Após a morte de Lênin, em homenagem ao grande líder o nome da cidade mudou novamente para Leningrado em 1924, e assim permaneceu até o fim da União Soviética. Em 1991, a cidade voltou a ter o nome antigo. Durante a segunda guerra mundial a cidade de Leningrado sofreu um cerco das tropas alemãs que durou cerca de 900 dias. Foi uma época terrível, de muita fome e perdas humanas, que terminou em 27 de janeiro de 1944 com a vitória dos soviéticos. É considerada uma das vitórias mais custosas da história em termos de vidas humanas perdidas.

No auge da ofensiva alemã, no ano de 1942, vários atiradores de elite foram espalhados pela cidade, inclusive, em alguns pontos estratégicos da cidade mais de um atirador aguardavam soldados inimigos. A espionagem russa tinha informações detalhadas das habilidades desses atiradores, mas seus esconderijos eram excelentes, tornando a tarefa de um soldado soviético que desejasse cruzar a cidade extremamente difícil.

Problema

Os soldados soviéticos eram bem treinados, mas com o passar do tempo e a continuação do cerco à cidade, os melhores soldados foram sendo dizimados, uma vez que se errassem o alvo na primeira tentativa certamente eram mortos pelos soldados alemães na tocaia.

Sabendo a probabilidade de um soldado em matar um atirador alemão e sabendo também o número de balas que ele tinha à sua disposição, desejamos saber a probabilidade desse soldado conseguir chegar a um ponto estratégico de destino, partindo de um ponto estratégico de origem. O soldado, sendo muito experiente, sempre usava um caminho que maximizava a probabilidade de sucesso. Note que o soldado deve matar todos os atiradores presentes no caminho usado, inclusive os que estiverem nos pontos estratégicos de origem e destino.

Entrada e saída

Entrada

A entrada é composta por diversas instâncias e termina com final de arquivo (EOF). A primeira linha de cada instância contém 3 inteiros, N $(2 \leq N \leq 1000)$, M, e K $(0 \leq K \leq 1000)$ e a probabilidade P $(0 \leq P \leq 1)$ do soldado matar um atirador. Os inteiros N, M, e K representam respectivamente os números de pontos estratégicos, estradas ligando pontos estratégicos e balas carregadas pelo soldado soviético. Os pontos estratégicos são numerados de 1 a N.

Cada uma das próximas M linhas contém um par de inteiros i e j indicando que existe uma estrada ligando o ponto i ao j. Em seguida tem uma linha contendo um inteiro A $(0 \leq A \leq 2000)$, correspondendo ao número de atiradores na cidade, seguido por A inteiros indicando a posição de cada atirador. A última linha de cada instância contém dois inteiros indicando o ponto de partida e o destino do soldado.

Entrada e saída

Saída

Para cada instância imprima uma linha contendo a probabilidade de sucesso do soldado soviético. A probabilidade deve ser impressa com 3 casas decimais.

Exemplo de entradas e saídas

Exemplo de Entrada

```
3 2 10 0.1
1 2
2 3
10 1 1 3 3 1 3 1 1 3 3
1 3
5 5 10 0.3
1 2
2 4
2 5
4 5
5 3
6 3 3 3 3 3 3
1 3
```

Exemplo de Saída

```
0.000
0.001
```

Solução com complexidade $O(TVE)$

- Observe que, para maximizar a probabilidade de sucesso, o soldado deve encontrar o mínimo de atiradores possível
- Desta forma, o problema se reduz a se determinar o caminho mínimo entre o ponto de partida s e o ponto de chegada e
- Como o número de vértices é pequeno, é possível usar o algoritmo de Bellman-Ford
- Porém é necessário tomar dois cuidados
- O primeiro deles é que a distância inicial de s a s não é zero, e sim o número de soldados em s
- Em segundo lugar é preciso acelerar o algoritmo interompendo o laço caso não exista nenhuma atualização ao final de uma iteração
- Por fim, se o número de soldados no caminho mínimo for maior do que K, não há chance de sucesso

Solução com complexidade $O(TVE)$

```cpp
#include <bits/stdc++.h>

using namespace std;

struct Edge { int u, v; };

const int MAX { 1010 }, oo { 1000000010 };
int dist[MAX], soldiers[MAX];

double
solve(int s, int e, int N, int K, double P, const vector<Edge>& edges)
{
    for (int i = 1; i <= N; ++i)
        dist[i] = oo;

    dist[s] = soldiers[s];

    for (int i = 1; i <= N - 1; i++)
    {
        bool updated = false;

```

Solução com complexidade $O(TVE)$

```
22          for (const auto& edge : edges)
23              if (dist[edge.v] > dist[edge.u] + soldiers[edge.v])
24              {
25                  dist[edge.v] = dist[edge.u] + soldiers[edge.v];
26                  updated = true;
27              }
28
29          if (not updated)
30              break;
31      }
32
33      if (dist[e] > K)
34          return 0.0;
35
36      return pow(P, dist[e]);
37  }
38
39  int main()
40  {
41      ios::sync_with_stdio(false);
42
```

Solução com complexidade $O(TVE)$

```
43      int N, M, K;
44      double P;
45
46      while (cin >> N >> M >> K >> P)
47      {
48          memset(soldiers, 0, sizeof soldiers);
49
50          vector<Edge> edges;
51
52          while (M--)
53          {
54              int u, v;
55              cin >> u >> v;
56
57              edges.push_back(Edge { u, v });
58              edges.push_back(Edge { v, u });
59          }
60
61          int A;
62          cin >> A;
63
```

Solução com complexidade $O(TVE)$

```
64          while (A--)
65          {
66              int pos;
67              cin >> pos;
68
69              soldiers[pos]++;
70          }
71
72          int s, e;
73          cin >> s >> e;
74
75          auto ans = solve(s, e, N, K, P, edges);
76
77          cout.precision(3);
78          cout << fixed << ans << '\n';
79      }
80
81      return 0;
82  }
```

Solução alternativa SPFA complexidade $O(TVE)$

```cpp
#include <bits/stdc++.h>

using namespace std;

const int MAX { 1010 }, oo { 1000000010 };
int dist[MAX], soldiers[MAX];
vector<int> adj[MAX];

double
solve(int s, int e, int N, int K, double P)
{
    bitset<MAX> in_queue;

    for (int i = 1; i <= N; ++i)
        dist[i] = oo;

    dist[s] = soldiers[s];

    queue<int> q;
    q.push(s);
    in_queue[s] = true;
```

Solução alternativa SPFA complexidade $O(TVE)$

```
23      while (not q.empty())
24      {
25          auto u = q.front(); q.pop();
26          in_queue[u] = false;
27
28          for (const auto& v : adj[u])
29          {
30              auto w = soldiers[v];
31
32              if (dist[v] > dist[u] + w)
33              {
34                  dist[v] = dist[u] + w;
35
36                  if (not in_queue[v])
37                  {
38                      q.push(v);
39                      in_queue[v] = true;
40                  }
41              }
42          }
43      }
```

Solução alternativa SPFA complexidade $O(TVE)$

```
45      if (dist[e] > K)
46          return 0.0;
47
48      return pow(P, dist[e]);
49  }
50
51  int main()
52  {
53      ios::sync_with_stdio(false);
54
55      int N, M, K;
56      double P;
57
58      while (cin >> N >> M >> K >> P)
59      {
60          memset(soldiers, 0, sizeof soldiers);
61
62          for (int i = 1; i <= N; ++i)
63              adj[i].clear();
64
```

Solução alternativa SPFA complexidade $O(TVE)$

```
65          while (M--)
66          {
67              int u, v;
68              cin >> u >> v;
69
70              adj[u].push_back(v);
71              adj[v].push_back(u);
72          }
73
74          int A;
75          cin >> A;
76
77          while (A--)
78          {
79              int pos;
80              cin >> pos;
81
82              soldiers[pos]++;
83          }
84
```

Solução alternativa SPFA complexidade $O(TVE)$

```
85          int s, e;
86          cin >> s >> e;
87
88          auto ans = solve(s, e, N, K, P);
89
90          cout.precision(3);
91          cout << fixed << ans << '\n';
92      }
93
94      return 0;
95  }
```

Referências

1. UVA 10959 – The Party, Part I
2. URI 1205 – Cerco a Leningrado