Optimizacijske metode (IŠRM), 2. domača naloga

Anton Luka Šijanec

5. junij 2025

Povzetek

Rešitve 2. domače naloge (navodila: https://ucilnica.fmf.uni-lj.si/mod/resource/view.php?id=74468).

Rešitev

1. Privzemimo, da $uv \notin EG$, sicer naloga ni rešljiva. Rešitev:

   (a) Iz vhodnega grafa g izdelamo nov graf G':

      i. Za vsako vozlišče k iz vhodnega grafa, ki ni niti začetno niti končno, v nov graf dodamo dve vozlišči, k' in k'' in povezavo $k' \to k''$ s kapaciteto 1.

      ii. V nov graf dodamo vozlišči u' in v'.

      iii. Za vsako vozlišče k, ki je v vhodnem grafu povezano z vozliščem u, dodamo v novem grafu povezavo $u' \to k'$ s kapaciteto ∞.

      iv. Za vsako vozlišče k, ki je v vhodnem grafu povezano z vozliščem v, dodamo v nov graf povezavo $k'' \to v'$ s kapaciteto ∞.

      v. Za vsako povezavo $\{k, l\}$ v vhodnem grafu, ki ne vsebuje niti u niti v, dodamo v nov graf povezavi $k'' \to l'$ in $l'' \to k'$ s kapacitetama ∞.

   Na novem grafu najdimo prerez S, T z najmanjšo kapaciteto. Označimo množico „prereznih povezav“ E kot množico, za katero velja $\forall i'i'' \in EG' : i' \in S \wedge i'' \in T \Leftrightarrow i'i'' \in E$. Množica vozlišč, ki jih je treba odstraniti iz G, da u in v ne bosta več povezana, je $Z = \{i; i'i'' \in E\}$.

   Intuitiven dokaz veljavnosti

   - V $G \setminus Z$ ne obstaja $u, v-$pot. Če bi obstajala, je to povečujoča pot za algoritem Ford-Fulkerson, s katerim iščemo prerez z najmanjšo kapaciteto, torej S, T ne bi bil najmanjši prerez.
   - Nobeno vozlišče ni v Z po nepotrebnem. Za $z \in Z$ velja, da v $G \setminus (Z \setminus \{z\})$ obstaja $u, v-$pot, sicer S, T ni najmanjši prerez.

   (b) Za lažjo obdelavo preimenujmo imena vozlišč v številke tako, da vozlišči, ki ju želimo ločiti, preimenujemo v najmanjšo in največjo številko: $a \to 0, b \to 1, c \to 2, d \to 3, e \to 10, f \to 5, g \to 6, h \to 7, i \to 8, j \to 9, k \to 4$.

   Vhodna datoteka:

```
0 1
0 2
0 3
0 6
0 9
1 2
1 10
1 6
2 10
3 10
3 6
3 9
```

```
10 5
10 6
10 7
10 4
5 6
5 4
6 7
6 8
7 8
7 4
8 9
8 4
9 4
```

Program `prerezna_vozlišča.py` izdela nov graf iz vhodnega grafa:

```
#!/usr/bin/python3
import sys
povezave = []
inf = 9999999
maxvozl = 0
for line in sys.stdin:
        a, b = map(int, line.split(" "))
        povezave.append((min(a, b), max(a, b)))
        maxvozl = max(maxvozl, max(a, b))
print(2*maxvozl+1)
for i in range(1, maxvozl):
        print(f"{2*i}\t{2*i+1}\t1") # vhodne so sode, izhodne so lihe,
            izjema sta 0 in 2*maxvozl, vozlišče 1 ne obstaja, 2*maxvozl+1
            tudi ne
for a, b in povezave:
        if a == 0:
                print(f"0\t{b*2}\t{inf}")
                continue
        print(f"{a*2+1}\t{b*2}\t{inf}")
        if b == maxvozl:
                continue
        print(f"{b*2+1}\t{a*2}\t{inf}")
```

Program `najmanjši_prerez.cpp` najde množico „prereznih povezav“, torej takih povezav D, da če je S, T najmanjši prerez, velja $\forall s \in S, t \in T : st \in D \Leftrightarrow s \in S \wedge t \in T$:

```
#include <iostream>
#include <vector>
#include <queue>
#include <limits>
using namespace std;
class Oznaka {
        public:
                bool znak; // true za +, false za -
                int povezava;
                int pretok;
        Oznaka(bool znak, int povezava, int pretok) : znak(znak), povezava(
            povezava), pretok(pretok) {}
};
class Povezava {
        public:
                int pretok;
                int kapaciteta;
```

```
        int od;
        int k;
    Povezava(int pretok, int kapaciteta, int od, int k) : pretok(pretok
        ), kapaciteta(kapaciteta), od(od), k(k) {}
};
int main (void) {
    int n;
    cin >> n;
    vector<vector<int>> izhodne;
    vector<vector<int>> vhodne;
    izhodne.resize(n);
    vhodne.resize(n);
    vector<Povezava> povezave;
    for (int a; cin >> a;) {
        int b, c;
        cin >> b >> c;
        if (c <= 0)
            continue;
        izhodne[a].push_back(povezave.size());
        vhodne[b].push_back(povezave.size());
        povezave.push_back(Povezava(0, c, a, b));
    }
    vector<Oznaka> oznake;
    oznake.reserve(n);
    for (int i = 0; i < n; i++)
        oznake.push_back(Oznaka(true, -1, -1));
    queue<int> obišči;
    oznake[0].pretok = numeric_limits<int>::max();
    obišči.push(0);
    while (!obišči.empty()) {
        auto vozlišče = obišči.front();
        obišči.pop();
        // cerr << vozlišče << endl;
        if (vozlišče == n-1) {
            // cout << oznake[vozlišče].pretok << ": ";
            for (int i = vozlišče; i != 0; oznake[i].znak ? i =
                povezave[oznake[i].povezava].od : i = povezave[
                oznake[i].povezava].k) {
                // cout << i << (oznake[i].znak ? "+" :
                    "-") << " ";
                povezave[oznake[i].povezava].pretok += (
                    oznake[i].znak ? 1 : -1) * oznake[vozliš
                    če].pretok;
            }
            // cout << "0" << endl;
            for (int i = 0; i < n; i++) {
                oznake[i].znak = true;
                oznake[i].povezava = -1;
                oznake[i].pretok = -1;
            }
            oznake[0].pretok = numeric_limits<int>::max();
            while (!obišči.empty())
                obišči.pop();
            obišči.push(0);
            continue;
        }
```

```
            for (auto povezava : izhodne[vozlišče]) {
                    // cerr << "izhodna: " << povezava << endl;
                    auto sosed = povezave[povezava].k;
                    if (oznake[sosed].pretok != -1) // že označen
                            continue;
                    oznake[sosed].znak = true;
                    oznake[sosed].povezava = povezava;
                    oznake[sosed].pretok = min(oznake[vozlišče].pretok,
                         povezave[povezava].kapaciteta-povezave[povezava
                        ].pretok);
                    // cerr << "pretok: " << oznake[sosed].pretok <<
                        endl;
                    if (oznake[sosed].pretok == 0)
                            oznake[sosed].pretok = -1; // ne označimo,
                                če v resnici ni na povečujoči poti
                    else
                            obišči.push(sosed);
            }
            for (auto povezava : vhodne[vozlišče]) {
                    // cerr << "vhodna: " << povezava << endl;
                    auto sosed = povezave[povezava].od;
                    if (oznake[sosed].pretok != -1)
                            continue;
                    oznake[sosed].znak = false;
                    oznake[sosed].povezava = povezava;
                    oznake[sosed].pretok = min(oznake[vozlišče].pretok,
                         povezave[povezava].pretok);
                    if (oznake[sosed].pretok == 0)
                            oznake[sosed].pretok = -1;
                    else
                            obišči.push(sosed);
            }
    }
    obišči.push(0);
    vector<bool> obiskan; // vozlišča
    obiskan.resize(n);
    while (!obišči.empty()) {
            int vozlišče = obišči.front();
            obišči.pop();
            if (obiskan[vozlišče])
                    continue;
            obiskan[vozlišče] = true;
            for (int povezava : izhodne[vozlišče]) {
                    if (povezave[povezava].pretok < povezave[povezava].
                        kapaciteta)
                            obišči.push(povezave[povezava].k);
            }
            for (int povezava : vhodne[vozlišče]) {
                    if (povezave[povezava].pretok != 0)
                            obišči.push(povezave[povezava].od);
            }
    }
    for (unsigned povezava = 0; povezava < povezave.size(); povezava++)
            if (obiskan[povezave[povezava].od] && !obiskan[povezave[
                povezava].k])
                    cout << povezava << "\t" << povezave[povezava].od
```

```
                                << "\t" << povezave[povezava].k << "\t" <<
                                povezave[povezava].kapaciteta << endl;
        /* for (unsigned int povezava = 0; povezava < povezave.size();
            povezava++) {
                if (povezave[povezava].pretok > 0)
                        cerr << "debug" << "\t" << povezava << "\t" <<
                            povezave[povezava].od << "\t" << povezave[
                            povezava].k << "\t" << "\t" << povezave[povezava
                            ].pretok << "/" << povezave[povezava].kapaciteta
                             << endl;
        } */
}
```

Program `obdelaj_izhod_prerezna_vozlišča.sh` pretvori prerezne povezave iz novega grafa v vozlišča v starem grafu:

```
#!/bin/bash
while read line
do
        echo $((`cut -f2 <<<$line`/2))
done
```

Program `makefile` poskrbi, da se vsi zahtevani programi poženejo:

```
all: naloga1_rešitev.txt

najmanjši_prerez: najmanjši_prerez.cpp
        c++ -ggdb3 -Wall -Wextra -pedantic najmanjši_prerez.cpp -onajmanjš
            i_prerez

prerezna_vozlišča_vhod.txt: prerezna_vozlišča.py naloga1.txt
        ./prerezna_vozlišča.py < naloga1.txt > prerezna_vozlišča_vhod.txt

prerezna_vozlišča_izhod.txt: prerezna_vozlišča_vhod.txt najmanjši_prerez
        ./najmanjši_prerez < prerezna_vozlišča_vhod.txt > prerezna_vozlišč
            a_izhod.txt

naloga1_rešitev.txt: prerezna_vozlišča_izhod.txt
    obdelaj_izhod_prerezna_vozlišča.sh
        ./obdelaj_izhod_prerezna_vozlišča.sh < prerezna_vozlišča_izhod.txt
            > naloga1_rešitev.txt

.PHONY: clean
clean:
        rm -f najmanjši_prerez prerezna_vozlišča_vhod.txt prerezna_vozlišč
            a_izhod.txt naloga1_rešitev.txt
```

Rešitev naloge so naslednja vozlišča:

1
2
3
6
9

Vozlišča, ki jih je treba odstraniti, so b, c, d, g in j.

2. Rešitev:

   (a) Izračunamo vse možne razdalje med točkami, odstranimo duplikate in jih uredimo po vrsti. Natanko ena izmed teh vrednosti bo dolžina najdaljše povezave v rešitvi. Za vsako vrednost i izdelamo graf G_i, ki se

od vhodnega grafa G razlikuje po tem, da mu odstranimo povezave, daljše od i. Na vsakem grafu G_i najdemo največje prirejanje. Rešitev je popolno prirejanje grafa G_i, kjer je i najmanjši. Seveda popolno prirejanje ne obstaja nujno za vsak G_i, vendar očitno obstaja vsaj za en G_i. Iskanje ustreznega i lahko pospešimo tako, da na urejenem seznamu dolžin delamo bisekcijo.

Časovna zahtevnost: $O\left((2n)^{2,5}\log n^2\right) = O\left(n^{2,5}\log n\right)$, ker je $O\left(n^{2,5}\right)$ zahtevnost algoritma Hopcroft–Karp za iskanje največjega prirejanja, $O(\log n)$ pa zahtevnost bisekcije.

(b) Vhodna datoteka:

```
0 1
0 2
0 3
0 6
0 9
1 2
1 10
1 6
2 10
3 10
3 6
3 9
10 5
10 6
10 7
10 4
5 6
5 4
6 7
6 8
7 8
7 4
8 9
8 4
9 4
```

Program, ki izpljune rešitev:

```
#include <iostream> // c++ -ggdb3 -Wall -Wextra -pedantic naloga2.cpp -onaloga2
#include <vector>
#include <algorithm>
#include <queue>
#include <limits>
#include <cassert>
using namespace std;
int main (void) {
        vector<pair<int, int>> točke;
        int a;
        while (cin >> a) {
                int b;
                cin >> b;
                točke.push_back(make_pair(a, b));
        }
        int n = točke.size()/2;
        vector<int> razdalje;
#define KVADRAT(x) ((x)*(x))
#define RAZDALJA(i, j) (KVADRAT(točke[(i)].first-točke[n+(j)].first)+KVADRAT(točke[(i)].second-točke[n+(j)].second)) // RAZDALJA(zgornji, spodnji)
```

```
        razdalje.reserve(KVADRAT(n));
        for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)
                        razdalje.push_back(RAZDALJA(i, j));
        sort(razdalje.begin(), razdalje.end());
        int l = n;
        int r = KVADRAT(n)-1;
        int minuspelo = numeric_limits<int>::max();
        vector<int> minmatching;
        while (l != r) {
                int m = (l+r)/2;
                vector<int> matching;
                vector<int> matching_reverse;
                matching.resize(n, -1);
                matching_reverse.resize(n, -1);
                while (true) {
išči_povečujočo_pot:
                        int prosto = 0;
                        for (; prosto < n && matching[prosto] != -1; prosto
                            ++);
                        if (prosto >= n) {
                                if (m < minuspelo) {
                                        minuspelo = m;
                                        minmatching = matching;
                                }
                                r = m-1;
                                break;
                        }
                        /* for (int i = 0; i < n; i++) {
                                if (RAZDALJA(prosto, i) > razdalje[m])
                                        continue;
                                if (matching_reverse[i] == -1) {
                                        matching_reverse[i] = prosto;
                                        matching[prosto] = i;
                                        goto išči_povečujočo_pot;
                                }
                        } */
                        for (int i = 0; i < n; i++) {
                                if (RAZDALJA(prosto, i) > razdalje[m])
                                        continue;
                                vector<bool> obiskan; // za zgornje
                                obiskan.resize(n);
                                vector<int> predhodnik; // za spodnje
                                predhodnik.resize(n, -1);
                                predhodnik[i] = prosto;
                                queue<int> bfs;
                                bfs.push(i);
                                while (!bfs.empty()) {
                                        int vozlišče = bfs.front();
                                        bfs.pop();
                                        if (matching_reverse[vozlišče] ==
                                            -1) {
                                                int v = vozlišče;
                                                int len = 0;
                                                while (predhodnik[v] !=
                                                    prosto) {
```

```
                                        len++;
                                        matching_reverse[v]
                                            = predhodnik[v
                                            ];
                                        int newv = matching
                                            [predhodnik[v]];
                                        matching[predhodnik
                                            [v]] = v;
                                        v = newv;
                                    };
                                    matching[prosto] = v;
                                    matching_reverse[v] =
                                        prosto;
                                    // cerr << "našel povečujoč
                                        o pot dolžine " << len
                                        << endl;
                                    for (int j = 0; j < n; j++)
                                         {
                                        assert(matching[j]
                                            == -1 ||
                                            matching_reverse
                                            [matching[j]] ==
                                             j);
                                        assert(
                                            matching_reverse
                                            [j] == -1 ||
                                            matching[
                                            matching_reverse
                                            [j]] == j);
                                    }
                                    goto išči_povečujočo_pot;
                                }
                                for (int j = 0; j < n; j++) {
                                    if (RAZDALJA(
                                        matching_reverse[vozlišč
                                        e], j) > razdalje[m] ||
                                        predhodnik[j] != -1 || (
                                        matching_reverse[j] !=
                                        -1 && obiskan[
                                        matching_reverse[j]]))
                                            continue;
                                    obiskan[matching_reverse[
                                        vozlišče]] = true;
                                    predhodnik[j] =
                                        matching_reverse[vozlišč
                                        e];
                                    bfs.push(j);
                                }
                            }
                        }
                        l = m+1;
                        break;
                    }
                }
                cout << minuspelo << "\t" << razdalje[minuspelo] << endl;
                cout << "i:";
```

```
        for (int i = 0; i < n; i++)
                cout << "\t" << i << "@(" << točke[i].first << "," << točke
                    [i].second << ")";
        cout << endl;
        cout << "m:";
        for (int i = 0; i < n; i++)
                cout << "\t" << minmatching[i] << "@(" << točke[minmatching
                    [i]+n].first << "," << točke[minmatching[i]+n].second <<
                     ")";
        cout << endl;
        cout << "r:";
        for (int i = 0; i < n; i++)
                cout << "\t" << RAZDALJA(i, minmatching[i]);
        cout << endl;
}
```

Izhod programa:

40	29						
i:	0@(7,9)	1@(9,9)	2@(4,6)	3@(7,5)	4@(7,8)	5@(3,1)	6@(9,4)	7@(2,6)
m:	3@(4,8)	4@(6,9)	7@(8,4)	6@(8,6)	5@(7,8)	1@(5,6)	2@(9,0)	0@(1,10)
r:	10	9	20	2	0	29	16	17

Prirejanje z najdaljšo dolžino $\sqrt{29}$ je $M = \{(0,3),(1,4),(2,7),(3,6),(4,5),(5,1),(6,2),(7,0)\}$.

Predhodne in končne določbe

Za večjo berljivost so programi in vse ostale datoteke, povezane s to domačo nalogo, objavljene na internetu na http://ni.šijanec.eu./sijanec/r/tree/šola/om/dn2/.