**LAPORAN TUGAS 1**

**Praktikum Analisis Algoritma**

Algoritma Stable Matching Problem

****

140810160014

SACHI HONGO

KELAS B

S-1 TEKNIK INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

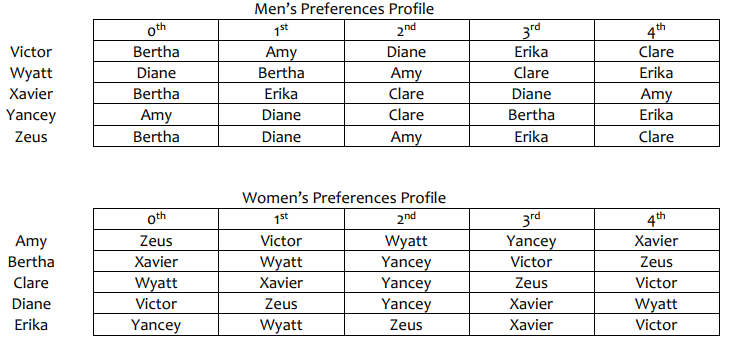
UNIVERSITAS PADJADJARAN

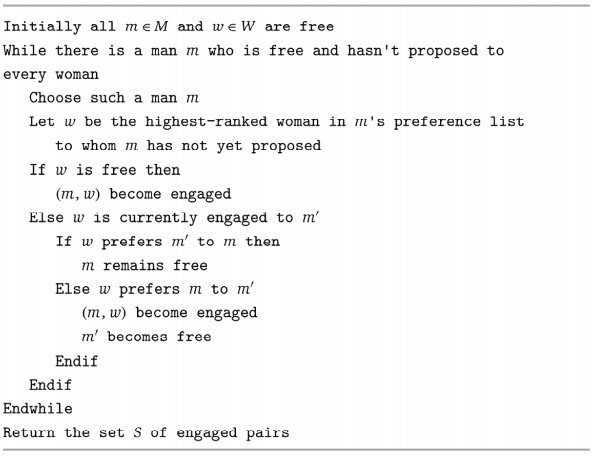
2018/2019

**Tugas Praktikum**

**Worksheet 1**

Dengan Algoritma Gale-Shapley, cari himpunan stable-matching yang sesuai dengan preference-list berikut ini. Gunakan processor terhebat yang anda miliki (otak) untuk mengikuti algoritma G-S dan output tidak perlu diuraikan per-looping tetapi Anda harus memahami hasil setiap looping.





/\*\*

\* Nama : Sachi Hongo

\* NPM : 140810160014

\* Kelas : B

\* Deskripsi : Implementasi Algoritma Gale Shapley pada Bahasa Java

\*\*/

/\*\* Class GaleShapley \*\*/

public class GaleShapley

{

private int N, engagedCount;

private String[][] menPref;

private String[][] womenPref;

private String[] men;

private String[] women;

private String[] womenPartner;

private boolean[] menEngaged;

/\*\* Constructor \*\*/

public GaleShapley(String[] m, String[] w, String[][] mp, String[][] wp)

{

N = mp.length;

engagedCount = 0;

men = m;

women = w;

menPref = mp;

womenPref = wp;

menEngaged = new boolean[N];

womenPartner = new String[N];

calcMatches();

}

/\*\* function to calculate all matches \*\*/

private void calcMatches()

{

while (engagedCount < N)

{

int free;

for (free = 0; free < N; free++)

if (!menEngaged[free])

break;

for (int i = 0; i < N && !menEngaged[free]; i++)

{

int index = womenIndexOf(menPref[free][i]);

if (womenPartner[index] == null)

{

womenPartner[index] = men[free];

menEngaged[free] = true;

engagedCount++;

}

else

{

String currentPartner = womenPartner[index];

if (morePreference(currentPartner, men[free], index))

{

womenPartner[index] = men[free];

menEngaged[free] = true;

menEngaged[menIndexOf(currentPartner)] = false;

}

}

}

}

printCouples();

}

/\*\* function to check if women prefers new partner over old assigned partner \*\*/

private boolean morePreference(String curPartner, String newPartner, int index)

{

for (int i = 0; i < N; i++)

{

if (womenPref[index][i].equals(newPartner))

return true;

if (womenPref[index][i].equals(curPartner))

return false;

}

return false;

}

/\*\* get men index \*\*/

private int menIndexOf(String str)

{

for (int i = 0; i < N; i++)

if (men[i].equals(str))

return i;

return -1;

}

/\*\* get women index \*\*/

private int womenIndexOf(String str)

{

for (int i = 0; i < N; i++)

if (women[i].equals(str))

return i;

return -1;

}

/\*\* print couples \*\*/

public void printCouples()

{

System.out.println("Couples are : ");

for (int i = 0; i < N; i++)

{

System.out.println(womenPartner[i] +" "+ women[i]);

}

}

/\*\* main function \*\*/

public static void main(String[] args)

{

System.out.println("Gale Shapley Marriage Algorithm\n");

/\*\* list of men \*\*/

String[] m = {"Victor", "Wyatt", "Xavier", "Yancey", "Zeus"};

/\*\* list of women \*\*/

String[] w = {"Amy", "Bertha", "Clare", "Diane", "Erika"};

/\*\* men preference \*\*/

String[][] mp = {{"Bertha", "Amy", "Diane", "Erika", "Clare"},

{"Diane", "Bertha", "Amy", "Clare", "Erika"},

{"Bertha", "Erika", "Clare", "Diane", "Amy"},

{"Amy", "Diane", "Clare", "Bertha", "Erika"},

{"Bertha", "Diane", "Amy", "Erika", "Clare"}};

/\*\* women preference \*\*/

String[][] wp = {{"Zeus", "Victor", "Wyatt", "Yancey", "Xavier"},

{"Xavier", "Wyatt", "Yancey", "Victor", "Zeus"},

{"Wyatt", "Xavier", "Yancey", "Zeus", "Victor"},

{"Victor", "Zeus", "Yancey", "Xavier", "Wyatt"},

{"Yancey", "Wyatt", "Zeus", "Xavier", "Victor"}};

GaleShapley gs = new GaleShapley(m, w, mp, wp);

}

}

womenPartner = new String[N];

calcMatches();

}

/\*\* function to calculate all matches \*\*/

private void calcMatches()

{

while (engagedCount < N)

{

int free;

for (free = 0; free < N; free++)

if (!menEngaged[free])

break;

for (int i = 0; i < N && !menEngaged[free]; i++)

{

int index = womenIndexOf(menPref[free][i]);

if (womenPartner[index] == null)

{

womenPartner[index] = men[free];

menEngaged[free] = true;

engagedCount++;

}

else

{

String currentPartner = womenPartner[index];

if (morePreference(currentPartner, men[free], index))

{

womenPartner[index] = men[free];

menEngaged[free] = true;

menEngaged[menIndexOf(currentPartner)] = false;

}

}

}

}

printCouples();

}

/\*\* function to check if women prefers new partner over old assigned partner \*\*/

private boolean morePreference(String curPartner, String newPartner, int index)

{

for (int i = 0; i < N; i++)

{

if (womenPref[index][i].equals(newPartner))

return true;

if (womenPref[index][i].equals(curPartner))

return false;

}

return false;

}

/\*\* get men index \*\*/

private int menIndexOf(String str)

{

for (int i = 0; i < N; i++)

if (men[i].equals(str))

return i;

return -1;

}

/\*\* get women index \*\*/

private int womenIndexOf(String str)

{

for (int i = 0; i < N; i++)

if (women[i].equals(str))

return i;

return -1;

}

/\*\* print couples \*\*/

public void printCouples()

{

System.out.println("Couples are : ");

for (int i = 0; i < N; i++)

{

System.out.println(womenPartner[i] +" "+ women[i]);

}

}

/\*\* main function \*\*/

public static void main(String[] args)

{

System.out.println("Gale Shapley Marriage Algorithm\n");

/\*\* list of men \*\*/

String[] m = {"Victor", "Wyatt", "Xavier", "Yancey", "Zeus"};

/\*\* list of women \*\*/

String[] w = {"Amy", "Bertha", "Clare", "Diane", "Erika"};

/\*\* men preference \*\*/

String[][] mp = {{"Bertha", "Amy", "Diane", "Erika", "Clare"},

{"Diane", "Bertha", "Amy", "Clare", "Erika"},

{"Bertha", "Erika", "Clare", "Diane", "Amy"},

{"Amy", "Diane", "Clare", "Bertha", "Erika"},

{"Bertha", "Diane", "Amy", "Erika", "Clare"}};

/\*\* women preference \*\*/

String[][] wp = {{"Zeus", "Victor", "Wyatt", "Yancey", "Xavier"},

{"Xavier", "Wyatt", "Yancey", "Victor", "Zeus"},

{"Wyatt", "Xavier", "Yancey", "Zeus", "Victor"},

{"Victor", "Zeus", "Yancey", "Xavier", "Wyatt"},

{"Yancey", "Wyatt", "Zeus", "Xavier", "Victor"}};

GaleShapley gs = new GaleShapley(m, w, mp, wp);

}

}

menEngaged[free] = true;

menEngaged[menIndexOf(currentPartner)] = false;

}

}

}

}

printCouples();

}

/\*\* function to check if women prefers new partner over old assigned partner \*\*/

private boolean morePreference(String curPartner, String newPartner, int index)

{

for (int i = 0; i < N; i++)

{

if (womenPref[index][i].equals(newPartner))

return true;

if (womenPref[index][i].equals(curPartner))

return false;

}

return false;

}

/\*\* get men index \*\*/

private int menIndexOf(String str)

{

for (int i = 0; i < N; i++)

if (men[i].equals(str))

return i;

return -1;

}

/\*\* get women index \*\*/

private int womenIndexOf(String str)

{

for (int i = 0; i < N; i++)

if (women[i].equals(str))

return i;

return -1;

}

/\*\* print couples \*\*/

public void printCouples()

{

System.out.println("Couples are : ");

for (int i = 0; i < N; i++)

{

System.out.println(womenPartner[i] +" "+ women[i]);

}

}

/\*\* main function \*\*/

public static void main(String[] args)

{

System.out.println("Gale Shapley Marriage Algorithm\n");

/\*\* list of men \*\*/

String[] m = {"Victor", "Wyatt", "Xavier", "Yancey", "Zeus"};

/\*\* list of women \*\*/

String[] w = {"Amy", "Bertha", "Clare", "Diane", "Erika"};

/\*\* men preference \*\*/

String[][] mp = {{"Bertha", "Amy", "Diane", "Erika", "Clare"},

{"Diane", "Bertha", "Amy", "Clare", "Erika"},

{"Bertha", "Erika", "Clare", "Diane", "Amy"},

{"Amy", "Diane", "Clare", "Bertha", "Erika"},

{"Bertha", "Diane", "Amy", "Erika", "Clare"}};

/\*\* women preference \*\*/

String[][] wp = {{"Zeus", "Victor", "Wyatt", "Yancey", "Xavier"},

{"Xavier", "Wyatt", "Yancey", "Victor", "Zeus"},

{"Wyatt", "Xavier", "Yancey", "Zeus", "Victor"},

{"Victor", "Zeus", "Yancey", "Xavier", "Wyatt"},

{"Yancey", "Wyatt", "Zeus", "Xavier", "Victor"}};

GaleShapley gs = new GaleShapley(m, w, mp, wp);

}

}

/\*\* get women index \*\*/

private int womenIndexOf(String str)

{

for (int i = 0; i < N; i++)

if (women[i].equals(str))

return i;

return -1;

}

/\*\* print couples \*\*/

public void printCouples()

{

System.out.println("Couples are : ");

for (int i = 0; i < N; i++)

{

System.out.println(womenPartner[i] +" "+ women[i]);

}

}

/\*\* main function \*\*/

public static void main(String[] args)

{

System.out.println("Gale Shapley Marriage Algorithm\n");

/\*\* list of men \*\*/

String[] m = {"Victor", "Wyatt", "Xavier", "Yancey", "Zeus"};

/\*\* list of women \*\*/

String[] w = {"Amy", "Bertha", "Clare", "Diane", "Erika"};

/\*\* men preference \*\*/

String[][] mp = {{"Bertha", "Amy", "Diane", "Erika", "Clare"},

{"Diane", "Bertha", "Amy", "Clare", "Erika"},

{"Bertha", "Erika", "Clare", "Diane", "Amy"},

{"Amy", "Diane", "Clare", "Bertha", "Erika"},

{"Bertha", "Diane", "Amy", "Erika", "Clare"}};

/\*\* women preference \*\*/

String[][] wp = {{"Zeus", "Victor", "Wyatt", "Yancey", "Xavier"},

{"Xavier", "Wyatt", "Yancey", "Victor", "Zeus"},

{"Wyatt", "Xavier", "Yancey", "Zeus", "Victor"},

{"Victor", "Zeus", "Yancey", "Xavier", "Wyatt"},

{"Yancey", "Wyatt", "Zeus", "Xavier", "Victor"}};

GaleShapley gs = new GaleShapley(m, w, mp, wp);

}

}

{"Diane", "Bertha", "Amy", "Clare", "Erika"},

{"Bertha", "Erika", "Clare", "Diane", "Amy"},

{"Amy", "Diane", "Clare", "Bertha", "Erika"},

{"Bertha", "Diane", "Amy", "Erika", "Clare"}};

/\*\* women preference \*\*/

String[][] wp = {{"Zeus", "Victor", "Wyatt", "Yancey", "Xavier"},

{"Xavier", "Wyatt", "Yancey", "Victor", "Zeus"},

{"Wyatt", "Xavier", "Yancey", "Zeus", "Victor"},

{"Victor", "Zeus", "Yancey", "Xavier", "Wyatt"},

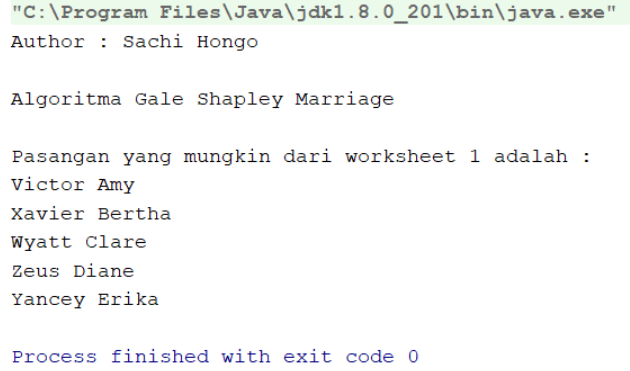
{"Yancey", "Wyatt", "Zeus", "Xavier", "Victor"}};

GaleShapley gs = new GaleShapley(m, w, mp, wp);

}

}

**Output**



**Analisis Algoritma**

1. Apakah jawaban Anda di Worksheet 01 dan Program sama persis? Jika Tidak? Kenapa?

Jawaban : Jawaban dengan perhitungan yang saya hitung pada worksheet 01 sama dengan program yang dibuat dengan java. Berikut pembuktian hasilnya :

Misalkan kita akan menguji pseudocode dengan tabel pria sebagai acuan

1. m = victor

victor 🡪 Bertha

if (Bertha == free) //True

**(Victor, Bertha)**

1. m = wyatt

wyatt 🡪 Diane

if (Diane == free) //true

**(wyatt, diane)**

1. m = Xavier

Xavier 🡪 Bertha

If (bertha == free) //false

Else

If (bertha prefer victor) //false

Else (bertha prefer Xavier) //true

**(Xavier, bertha)**

**Victor free**

1. m = Yancey

Yancey 🡪 Amy

If (Amy == free) //true

**(Yancey, Amy)**

1. m = Zeus

Zeus 🡪 Bertha

If (Bertha == free) //false

Else

If (bertha prefer Xavier) // true

**(Xavier, Bertha)**

**Zeus free**

1. m = Victor

Victor 🡪 Amy

If (Amy == free) //false

Else

If (amy prefer yancey) //false

Else (amy prefer victor) //true

**(Victor, Amy)**

**Yancey free**

1. m = Zeus

Zeus 🡪 Diane

If (Diane == free) //false

Else

If (Diane prefer wyatt) //false

Else (Diane prefer zeus) //true

**(Zeus, Diane)**

**Wyatt free**

1. m = Yancey

yancey 🡪 Diane

if (diane == free) //false

else

if (diane prefer zeus) //true

**(Zeus, Diane)**

**Yancey free**

1. m = wyatt

wyatt 🡪 bertha

if (bertha == free) // false

else

if (bertha prefer Xavier) //true

**(Xavier, Bertha)**

**Wyatt free**

1. m = yancey

yancey 🡪 clare

if (clare == free) //true

**(Yancey, Clare)**

1. m = wyatt

wyatt 🡪 Amy

if (amy == free) //false

else

if (amy prefer victor) //true

**(Victor, Amy)**

**Wyatt free**

1. m = wyatt

wyatt 🡪 Clare

if (clare == free) //false

else

if (clare prefer yancey) //false

else (clare prefer wyatt) //true

**(Wyatt, Clare)**

**Yancey free**

1. m = yancey

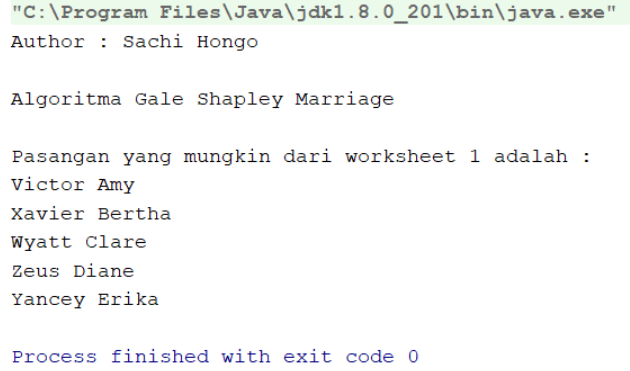
yancey 🡪 Erika

if (Erika == free) //true

**(Yancey, Erika)**

Jadi pasangannya :

* Yancey, Erika
* Wyatt, Clare
* Victor, Amy
* Xavier, bertha
* Zeus, Diane



Jika dilihat di atas, maka pengerjaan yang dilakukan berdasarkan pseudocode dan dengan program menghasilkan pasangan yang sama

**Fakta(1.1) :**

Seorang wanita tetap bertunangan dari titik di mana dia menerima proposal pertamanya; dan  
urutan mitra yang bertunangan dengannya menjadi lebih baik dan lebih baik lagi (hal ini sesuai dengan daftar preferensi wanita). 🡪 tidak perlu dipertanyakan

-> Pandangan seorang pria selama eksekusi algoritma agak berbeda. Dia bebas sampai dia melamar wanita dengan peringkat tertinggi di daftar; pada titik ini dia mungkin atau mungkin tidak bertunangan. Seiring berjalannya waktu, ia dapat bergantian antara bebas dan bertunangan; Namun, properti berikut tidak berlaku bagi pria dimana ia bisa saja diputuskan jika si wanita itu lebih memilih pria yang berada dalam list utamanya.

**Fakta (1.2) :**

Urutan wanita yang dilamar pria lebih buruk dan lebih buruk lagi (hal ini sesuai dengan daftar  
preferensi pria). 🡪 tidak perlu dipertanyakan

-> Ketika eksekusi algoritma akan berakhir, dan akan memberikan batasan pada jumlah iterasi maksimum yang diperlukan untuk pemutusan hubungan, oleh karena itu list wanita pada pria semakin menurun seiring jika pria itu belum berpasangan.

**Teorema (1.3) :**

Algoritma G-S berakhir setelah paling banyak n2 iterasi menggunakan While Loop.

-> Setiap kali menggunakan while loop, satu orang pria melamar satu orang perempuan. Sehingga, paling tidak ada n2 lamaram yang mungkin. Algoritma ini terus membuat kemajuan. Dalam setiap iterasi loop sementara, seorang pria bebas (belum berpasangan) yang melamar wanita berikutnya dalam daftar pilihannya, seseorang yang belum pernah ia ajukan sebelumnya. Karena ada n pria dan setiap daftar preferensi memiliki n panjang, ada sebagian besar proposal yang dapat terjadi. Jadi jumlah iterasi yang dapat terjadi paling banyak adalah n2. Selanjutnya membuktikan bahwa pencocokan yang dikembalikan stabil. Untuk melakukan itu, kita dapat melakukan dua pengamatan: yang pertama pada urutan pria yang bertunangan dengan wanita, dan yang kedua pada pria yang masih bebas. Contoh jika ada 3 pria dan 3 wanita maka kemungkinan berpasangannya ada 9 kemungkinan. Karena n=3, dan n2 adalah 9.

|  |  |  |  |
| --- | --- | --- | --- |
| Pria\Wanita | D | E | F |
| A | A,D | A,E | A,F |
| B | B,D | B,E | B,F |
| C | C,D | C,E | C,F |

**Teorema (1.4) :**

Jika seorang pria bebas di beberapa titik dalam eksekusi algoritma, maka ada seorang wanita  
yang belum dia ajak bertunangan.

-> Misalkan ada saatnya ketika m (pria) berstatus bebas, tetapi telah mengusulkan bertunangan untuk setiap wanita. Kemudian pada Fakta (1.1) diketahui bahwa, masing-masing dari n wanita memounyai peran penting dan terlibat pada saat ini. Karena set pasangan yang bertunangan membentuk pasangan yang cocok, maka harus ada juga n pria yang bertunangan pada saat ini. Tetapi hanya ada n pria total, dan m (pria) tidak bertunangan, jadi hal ini adalah sebuah kontradiksi. Yang ada dimana ketika seorang pria itu bebas di beberapa titik, itu artinya wanita yang berpasangan sebelumnya telah menemukan pasangan pria lain yang lebih cocok darinya dan pria tersebut harus mencoba wanita lain yang berada dalam listnya yang semakin mengerucut ke bawah.

**Teorema (1.5) :**

Himpunan S yang dikembalikan saat terminasi adalah *perfect matching*

-> Set pasangan yang bertunangan selalu membentuk pasangan yang cocok. Mari kita anggap bahwa algoritma berakhir dengan m pria bebas. Pada saat pemutusan hubungan, hal ini menjadi kasus bahwa m telah mengusulkan lamaran kepada setiap wanita, karena jika tidak, maka loop tidak akan terlepas. Tetapi ini bertentangan dengan teorema (1.4), yang mengatakan bahwa tidak mungkin ada pria bebas yang telah melamar setiap wanita. Pria pasti hanya akan melamar apabila belum atau pasangan sebelumnya tidak cocok. Wanita akan selalu memilih pria terbaik untuk bertunangan dengannya.

-> Dengan itu himpunan S adalah perfect matching sesuai asumsi di atas.

**Teorema (1.6) :**

Sebuah eksekusi algoritma G-S mengembalikan satu set pasangan S. Set S adalah pasangan yang stabil.

-> Kita telah melihat, pada teorema (1.5), bahwa S adalah pasangan yang cocok. Jadi, untuk membuktikan bahwa S adalah pasangan yang stabil, kita akan mengasumsikan bahwa ada ketidakstabilan sehubungan dengan S dan mendapatkan kontradiksi. Seperti yang didefinisikan sebelumnya, ketidakstabilan seperti itu akan melibatkan dua pasangan, (m, w) dan (m ′, w ′), dalam S dengan sifat-sifat yang

* *m* prefers *w*′ to *w*, dan
* *w*′ prefers *m* to *m*′

Dalam pelaksanaan algoritma yang menghasilkan S, proposal terakhir, menurut definisi, adalah wanita (w). Sekarang kita bertanya: Apakah saya mengusulkan untuk ′ pada beberapa titik sebelumnya dalam eksekusi ini? Jika tidak, maka w harus muncul lebih tinggi pada daftar preferensi m daripada w w, bertentangan dengan asumsi kami bahwa m lebih suka w ′ daripada w. Jika dia melakukannya, maka dia ditolak oleh w favor demi beberapa orang lain m ′ ′, yang lebih suka m. m ′ adalah mitra terakhir dari w ′, jadi m ′ ′ = m ′ atau, dengan (1.1), w ′ lebih memilih mitra terakhirnya m ′ daripada m ′ ′; baik cara ini bertentangan dengan asumsi kami bahwa w ′ lebih suka m daripada m ′.

-> Seorang wanita bebas tidak akan menolak proposal pria, yaitu, hanya wanita yang cocok yang dapat menolak proposal pria. Dengan demikian, sudah ditolak oleh semua wanita, maka semua wanita pasti sudah cocok. Namun, seorang wanita hanya dapat dicocokkan dengan paling banyak satu pria, menyiratkan bahwa jika gratis, maka paling banyak 1 wanita dicocokkan. dengan demikian, setidaknya salah satu harus tetap, bebas dan tidak dapat ditolak oleh semua wanita. Kedua, setiap iterasi dari loop sementara melibatkan tepat satu proposal. Perhatikan bahwa karena pria bergerak monoton di daftar preferensi mereka, tidak ada pria yang akan melamar wanita yang sama dua kali. Karena tidak ada pria yang bisa ditolak oleh setiap wanita, dalam kasus terburuk, seorang pria akan melamar semua wanita sebelum dicocokkan. Dengan demikian, jumlah iterasi dari loop sementara paling tidak sebelum algoritma berhenti, dan ketika berhenti, setiap pria dan wanita dicocokkan.

-> Maka S adalah pasangan yang stabil.