10

Peningkatan berulang

Orang yang paling sukses pada akhirnya adalah mereka yang keberhasilannya adalah hasil dari pertambahan yang mantap.

— Alexander Graham Bell (1835–1910)

Strategi serakah, yang dibahas dalam bab sebelumnya, membangun solusi untuk masalah optimasi sepotong demi sepotong, selalu menambahkan potongan optimal lokal ke solusi yang dibangun sebagian. Dalam bab ini, kita membahas pendekatan yang berbeda untuk merancang algoritma untuk masalah optimasi. Ini dimulai dengan beberapa solusi yang layak (solusi yang memenuhi semua kendala masalah) dan melanjutkan untuk memperbaikinya dengan aplikasi berulang dari beberapa langkah sederhana. Langkah ini biasanya melibatkan perubahan kecil yang terlokalisasi yang menghasilkan solusi yang layak dengan nilai fungsi tujuan yang ditingkatkan. Ketika tidak ada perubahan yang meningkatkan nilai fungsi tujuan, algoritme mengembalikan solusi fisibel terakhir sebagai optimal dan berhenti.

Ada beberapa kendala untuk keberhasilan implementasi ide ini. Pertama, kita membutuhkan solusi layak awal. Untuk beberapa masalah, kita selalu dapat memulai dengan solusi trivial atau menggunakan solusi perkiraan yang diperoleh dengan beberapa algoritma lain (misalnya, greedy). Tetapi bagi yang lain, menemukan solusi awal mungkin memerlukan upaya sebanyak memecahkan masalah setelah solusi yang layak telah diidentifikasi. Kedua, tidak selalu jelas perubahan apa yang harus diperbolehkan dalam solusi yang layak sehingga kita dapat memeriksa secara efisien apakah solusi saat ini optimal secara lokal dan, jika tidak, menggantinya dengan yang lebih baik. Ketiga—dan ini adalah kesulitan paling mendasar adalah masalah ekstrem lokal versus global (maksimum atau minimum). Pikirkan tentang masalah menemukan titik tertinggi di daerah perbukitan tanpa peta pada hari berkabut. Hal yang logis untuk dilakukan adalah mulai berjalan "menanjak" dari titik Anda berada sampai menjadi tidak mungkin untuk melakukannya karena tidak ada arah yang mengarah ke atas. Anda akan mencapai titik tertinggi lokal, tetapi karena kelayakan yang terbatas, tidak akan ada cara sederhana untuk mengetahui apakah titik tersebut adalah yang tertinggi (maksimum global yang Anda kejar) di seluruh area.

Untungnya, ada masalah penting yang dapat diselesaikan dengan algoritma perbaikan berulang. Yang paling penting dari mereka adalah pemrograman linier.

Kami telah menemukan topik ini di Bagian 6.6. Di sini, di Bagian 10.1, kami memperkenalkan metode simpleks, algoritma klasik untuk pemrograman linier. Ditemukan oleh ahli matematika AS George B. Dantzig pada tahun 1947, algoritme ini telah terbukti menjadi salah satu pencapaian paling penting dalam sejarah algoritme.

Dalam Bagian 10.2, kami mempertimbangkan masalah penting untuk memaksimalkan jumlah aliran yang dapat dikirim melalui jaringan dengan tautan dengan kapasitas terbatas. Masalah ini merupakan kasus khusus dari program linier. Namun, struktur khususnya memungkinkan untuk memecahkan masalah dengan algoritma yang lebih efisien daripada metode simpleks. Kami menguraikan algoritma perbaikan berulang klasik untuk masalah ini, ditemukan oleh matematikawan Amerika LR Ford, Jr., dan
DR Fulkerson pada 1950-an.

Dua bagian terakhir dari bab ini membahas tentang pencocokan bipartit. Ini adalah masalah menemukan pasangan optimal dari elemen yang diambil dari dua himpunan yang lepas. Contohnya termasuk pekerja dan pekerjaan yang cocok, lulusan sekolah menengah dan perguruan tinggi, dan pria dan wanita untuk menikah. Bagian 10.3 membahas masalah memaksimalkan jumlah pasangan yang cocok; Bagian 10.4 berkaitan dengan stabilitas pencocokan.

Kami juga membahas beberapa algoritme peningkatan iteratif di Bagian 12.3, di mana kami mempertimbangkan algoritme aproksimasi untuk masalah salesman dan knapsack keliling. Contoh lain dari algoritma perbaikan berulang dapat ditemukan di buku teks algoritma oleh Moret dan Shapiro [Mor91], buku tentang optimasi kontinu dan diskrit (misalnya, [Nem89]), dan literatur tentang pencarian heuristik (misalnya, [Mic10]).

10.1 **Metode Simpleks**

Kami telah menemukan pemrograman linier (lihat Bagian 6.6)—masalah umum dalam mengoptimalkan fungsi linier dari beberapa variabel tunduk pada satu set kendala linier:

memaksimalkan (atau meminimalkan) $C_1x_1 + \ldots + C_nx_nA_{Saya1}x_1 + \ldots + A_{di\ dalam}x$

tunduk pada $_n$ (atau ≥ atau =) $B_{Saya}x_1 \geq 0, \ldots, x_n \geq 0.$ untuk $Saya = 1, \ldots, M$

(10.1)

Kami menyebutkan di sana bahwa banyak masalah praktis yang penting dapat dimodelkan sebagai contoh program linier. Dua peneliti, LV Kantorovich dari bekas Uni Soviet dan TC Koopmans Belanda-Amerika, bahkan dianugerahi Hadiah Nobel pada tahun 1975 untuk kontribusi mereka pada teori pemrograman linier dan aplikasinya pada ekonomi. Rupanya karena tidak ada Hadiah Nobel dalam matematika, Akademi Ilmu Pengetahuan Kerajaan Swedia gagal menghormati ahli matematika AS GB Dantzig, yang secara universal diakui sebagai bapak linear

pemrograman dalam bentuk modern dan penemu metode simpleks, algoritma klasik untuk memecahkan masalah tersebut.[1]

Interpretasi Geometris dari Pemrograman Linier

Sebelum kita memperkenalkan metode umum untuk memecahkan masalah program linier, mari kita perhatikan contoh kecil, yang akan membantu kita untuk melihat sifat dasar dari masalah tersebut.

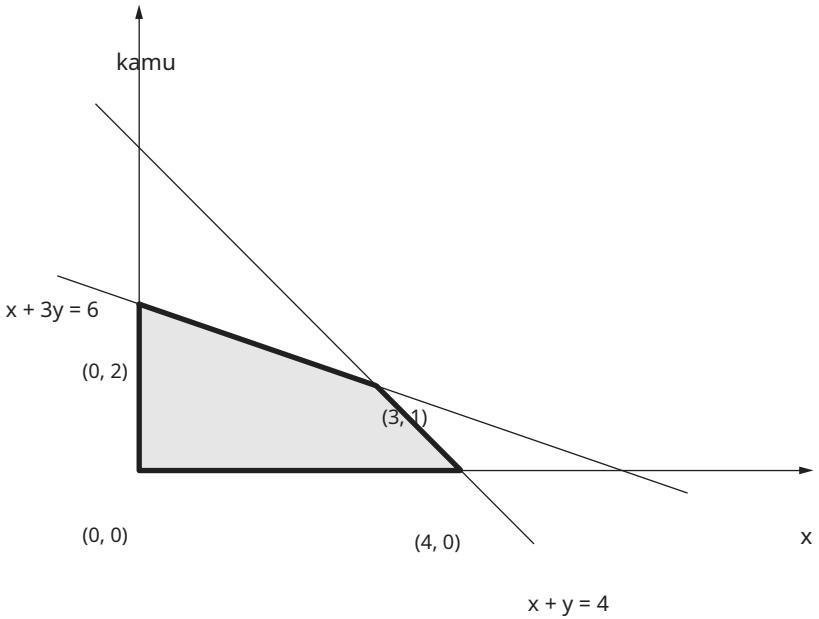
CONTOH 1 Perhatikan masalah program linier berikut: dalam dua variasi-mampu:

$$\begin{aligned} &\text{memaksimalkan} && 3x + 5kamu \\ &\text{tunduk pada} && x + kamu \le 4 \\ & && x + 3kamu \le 6 \\ & && x \ge 0,\ kamu \ge 0. \end{aligned} \tag{10.2}$$

Menurut definisi, A ***solusi yang layak*** untuk masalah ini ada gunanya (x, y) yang memuaskan semua kendala masalah; masalah ***wilayah yang layak*** adalah himpunan semua titik fisibelnya. Adalah instruktif untuk membuat sketsa wilayah yang layak di bidang Cartesian. Ingatlah bahwa setiap persamaan $kapak + oleh = C$, dimana koefisien A dan B keduanya tidak sama dengan nol, mendefinisikan garis lurus. Garis seperti itu membagi bidang menjadi dua setengah bidang: untuk semua titik di salah satunya, $kapak + oleh < c$, sedangkan untuk semua titik yang lain, $kapak + oleh > c$. (Sangat mudah untuk menentukan yang mana dari dua setengah bidang yang mana: ambil sembarang titik $(x_0, kamu_0)$ tidak di garis $kapak + oleh = C$ dan periksa mana dari dua ketidaksetaraan yang berlaku, $kapak_0 + oleh_0 > C$ atau $kapak_0 + oleh_0 < C.$) Secara khusus, himpunan titik yang didefinisikan oleh pertidaksamaan $x + kamu \le 4$ terdiri dari titik-titik di atas dan di bawah garis $x + kamu = 4$, dan himpunan titik-titik yang didefinisikan oleh pertidaksamaan $x + 3kamu \le 6$ terdiri dari titik-titik di atas dan di bawah garis $x + 3kamu = 6$. Karena titik-titik daerah fisibel harus memenuhi semua kendala masalah, daerah fisibel diperoleh dengan perpotongan dua setengah bidang ini dan kuadran pertama bidang kartesius yang ditentukan oleh kendala nonnegatif. $x \ge 0,\ kamu \ge 0$ (lihat Gambar 10.1). Dengan demikian, daerah layak untuk masalah (10.2) adalah poligon cembung dengan simpul $(0, 0), (4, 0), (0, 2)$, dan $(3, 1)$. (Titik terakhir, yang merupakan titik perpotongan garis $x + kamu = 4$ dan $x + 3kamu = 6$, diperoleh dengan menyelesaikan sistem dari dua persamaan linier ini.) Tugas kita adalah menemukan ***solusi optimal***, suatu titik pada daerah fisibel dengan nilai terbesar ***fungsi objektif*** $z = 3x + 5y$.

Apakah ada solusi fisibel yang nilai fungsi tujuannya sama dengan, katakanlah, 20? Intinya (x, y) dimana fungsi tujuan $z = 3x + 5kamu$ sama dengan 20 membentuk garis $3x + 5kamu = 20$. Karena garis ini tidak memiliki titik yang sama

1. George B. Dantzig (1914–2005) telah menerima banyak penghargaan, termasuk National Medal of Science yang dipersembahkan oleh presiden Amerika Serikat pada tahun 1976. Kutipan tersebut menyatakan bahwa National Medal dianugerahkan "untuk penemuan program linier dan metode penemuan yang mengarah pada aplikasi ilmiah dan teknis skala luas untuk masalah penting dalam logistik, penjadwalan, dan optimasi jaringan, dan penggunaan komputer dalam memanfaatkan teori matematika secara efisien."



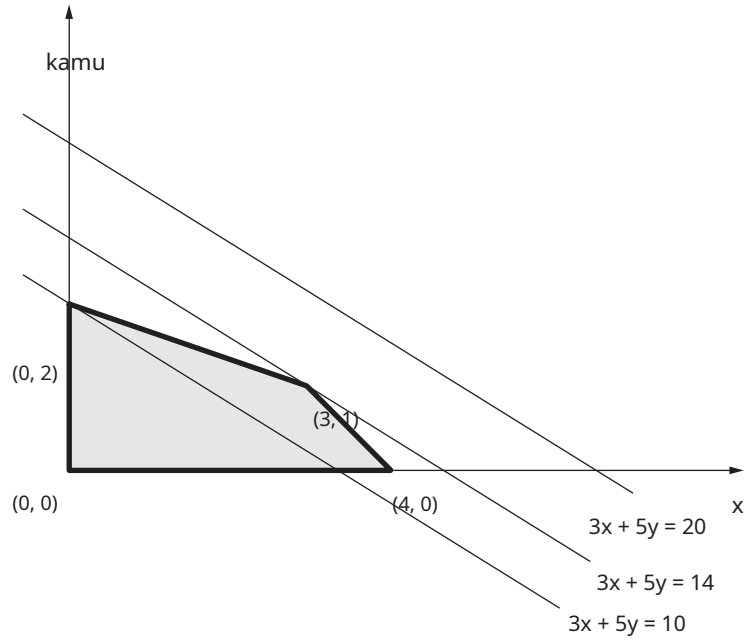
GAMBAR 10.1 Wilayah masalah yang layak (10.2).

dengan wilayah yang layak—lihat Gambar 10.2—jawaban atas pertanyaan yang diajukan adalah tidak. Di sisi lain, ada banyak titik layak yang fungsi tujuannya sama dengan, katakanlah, 10: mereka adalah titik persimpangan garis $3x + 5kamu = 10$ dengan daerah layak. Perhatikan bahwa garis $3x + 5kamu = 20$ dan $3x + 5kamu = 10$ memiliki kemiringan yang sama, seperti halnya garis yang didefinisikan oleh persamaan $3x + 5kamu = z$ di mana z adalah beberapa konstan. Garis seperti itu disebut ***garis level*** dari fungsi tujuan. Dengan demikian, masalah kita dapat dinyatakan kembali sebagai menemukan nilai terbesar dari parameter z untuk itu garis level $3x + 5kamu = z$ memiliki titik yang sama dengan daerah layak.

Kita dapat menemukan garis ini dengan menggeser, katakanlah, garis $3x + 5kamu = 20$ barat daya (tanpa mengubah kemiringannya!) menuju daerah yang layak sampai menyentuh daerah tersebut untuk pertama kalinya atau dengan menggeser, katakanlah, garis $3x + 5kamu = 10$ timur laut sampai menyentuh wilayah yang layak untuk terakhir kalinya. Bagaimanapun, itu akan terjadi pada intinya $(3, 1)$ dengan yang sesuai z nilai $3 \cdot 3 + 5 \cdot 1 = 14$. Ini berarti bahwa solusi optimal untuk masalah program linier yang dimaksud adalah $x = 3$, $kamu = 1$, dengan nilai maksimal fungsi tujuan sama dengan 14.

Perhatikan bahwa jika kita harus memaksimalkan $z = 3x + 3kamu$ sebagai fungsi tujuan dalam masalah (10.2), garis level $3x + 3kamu = z$ untuk nilai terbesar dari z akan bertepatan dengan segmen garis batas yang memiliki kemiringan yang sama dengan garis level (gambar garis ini pada Gambar 10.2). Akibatnya, semua titik segmen garis antara simpul $(3, 1)$ dan $(4, 0)$, termasuk simpul itu sendiri, akan menjadi solusi optimal, menghasilkan, tentu saja, nilai maksimal yang sama dari fungsi tujuan.

■

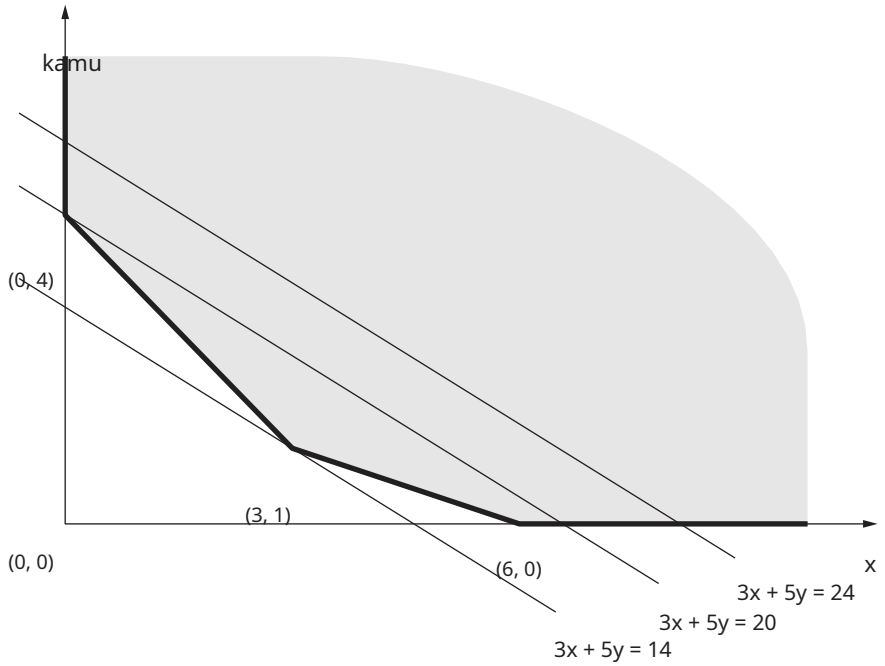


GAMBAR 10.2 Memecahkan masalah pemrograman linier dua dimensi secara geometris.

Apakah setiap masalah program linier memiliki solusi optimal yang dapat ditemukan pada titik dari daerah fisibelnya? Tanpa kualifikasi yang sesuai, jawaban untuk pertanyaan ini adalah tidak. Untuk memulainya, daerah fisibel dari masalah program linier bisa kosong. Misalnya, jika kendala mencakup dua persyaratan yang bertentangan, seperti: $x + kamu \le 1$ dan $x + kamu \ge 2$, tidak ada titik di wilayah layak masalah. Masalah pemrograman linier dengan daerah layak kosong disebut ***tidak mungkin***. Jelas, masalah yang tidak layak tidak memiliki solusi optimal.

Komplikasi lain mungkin muncul jika wilayah layak masalah tidak terbatas, seperti yang ditunjukkan contoh berikut.

CONTOH 2 Jika kita membalikkan pertidaksamaan dalam masalah (10.2) menjadi $x + kamu \ge 4$ dan $x + 3kamu \ge 6$, wilayah yang layak dari masalah baru akan menjadi tidak terbatas (lihat Gambar 10.3). Jika daerah fisibel dari masalah program linier tidak terbatas, fungsi tujuannya mungkin atau mungkin tidak mencapai nilai optimal yang terbatas di dalamnya. Misalnya, masalah memaksimalkan $z = 3x + 5kamu$ tunduk pada kendala $x + kamu \ge 4$, $x + 3kamu \ge 6$, $x \ge 0$, $kamu \ge 0$ tidak memiliki solusi optimal, karena terdapat titik-titik pada daerah fisibel yang membuat $3x + 5kamu$ sebesar yang kita inginkan. Masalah seperti itu disebut ***tak terbatas***. Di sisi lain, masalah meminimalkan $z = 3x + 5kamu$ tunduk pada kendala yang sama memiliki solusi optimal (yang?). ■



GAMBAR 10.3 Wilayah layak tak terbatas dari masalah pemrograman linier dengan kendala $x + kamu \geq 4$, $x + 3kamu \geq 6$, $x \geq 0$, $kamu \geq 0$, dan tiga garis level dari fungsi $3x + 5kamu$.

Untungnya, fitur paling penting dari contoh yang kami pertimbangkan di atas berlaku untuk masalah dengan lebih dari dua variabel. Secara khusus, daerah yang layak dari masalah pemrograman linier tipikal dalam banyak hal mirip dengan poligon cembung dalam bidang Cartesian dua dimensi. Secara khusus, selalu memiliki jumlah simpul yang terbatas, yang lebih disukai oleh matematikawan untuk disebut ***titik ekstrim*** (lihat Bagian 3.3). Selanjutnya, solusi optimal untuk masalah program linier dapat ditemukan pada salah satu titik ekstrim dari daerah fisibelnya. Kami mengulangi sifat-sifat ini dalam teorema berikut.

TEOREMA (*Teorema Titik Ekstrim*) Setiap masalah program linier dengan daerah layak terbatas tak kosong memiliki solusi optimal; Selain itu, solusi optimal selalu dapat ditemukan pada titik ekstrim dari kemungkinan masalah wilayah.[2]

Teorema ini mengimplikasikan bahwa untuk menyelesaikan masalah program linier, setidaknya dalam kasus daerah fisibel terbatas, kita dapat mengabaikan semua kecuali sejumlah terhingga.

2. Kecuali untuk beberapa contoh yang merosot (seperti memaksimalkan $z = x + kamu$ tunduk pada $x + kamu = 1$), jika masalah program linier dengan daerah fisibel tak terbatas memiliki solusi optimal, maka masalah tersebut juga dapat ditemukan pada titik ekstrim daerah fisibel.

titik di wilayah layak. Pada prinsipnya, kita dapat memecahkan masalah seperti itu dengan menghitung nilai fungsi tujuan pada setiap titik ekstrem dan memilih salah satu dengan nilai terbaik. Namun, ada dua kendala utama untuk mengimplementasikan rencana ini. Yang pertama terletak pada perlunya mekanisme untuk menghasilkan titik-titik ekstrem dari wilayah yang layak. Seperti yang akan kita lihat di bawah, prosedur aljabar yang agak langsung untuk tugas ini telah ditemukan. Kendala kedua terletak pada jumlah titik ekstrim yang dimiliki oleh suatu wilayah yang layak secara tipikal. Di sini, beritanya buruk: jumlah titik ekstrem diketahui tumbuh secara eksponensial dengan ukuran masalah. Hal ini membuat pemeriksaan menyeluruh dari titik-titik ekstrem tidak realistis untuk sebagian besar masalah pemrograman linier dengan ukuran nontrivial.

Untungnya, ternyata ada algoritma yang biasanya memeriksa hanya sebagian kecil dari titik ekstrem dari wilayah yang layak sebelum mencapai yang optimal. Algoritma terkenal ini disebut ***metode simpleks*** . Ide dari algoritma ini dapat digambarkan secara geometris sebagai berikut. Mulailah dengan mengidentifikasi titik ekstrem dari wilayah yang layak. Kemudian periksa apakah seseorang bisa mendapatkan nilai yang ditingkatkan dari fungsi tujuan dengan pergi ke titik ekstrim yang berdekatan. Jika tidak demikian, titik saat ini optimal—berhenti; jika demikian, lanjutkan ke titik ekstrem yang berdekatan dengan nilai fungsi tujuan yang ditingkatkan. Setelah sejumlah langkah yang terbatas, algoritma akan mencapai titik ekstrim di mana solusi optimal terjadi atau menentukan bahwa tidak ada solusi optimal.

Garis Besar Metode Simpleks

Tugas kita sekarang adalah "menerjemahkan" deskripsi geometrik metode simpleks ke dalam bahasa aljabar yang lebih tepat secara algoritmik. Untuk memulainya, sebelum kita dapat menerapkan metode simpleks pada masalah pemrograman linier, metode tersebut harus direpresentasikan dalam bentuk khusus yang disebut ***bentuk standar***. Bentuk standar memiliki persyaratan sebagai berikut:

- Ini harus menjadi masalah maksimalisasi.
- Semua kendala (kecuali kendala nonnegatif) harus dalam bentuk persamaan linier dengan sisi kanan nonnegatif.
- Semua variabel harus harus nonnegatif.

Jadi, regangan masalah program linier umum dan dalam bentuk standar dengan M menipu-n tidak diketahui $(n \geq M)$ adalah

$$\begin{aligned} &\text{memaksimalkan} && C_1x_1 + \ldots + C_nx_nA_{Saya1}x_1 + \ldots + A_{di\ dalam}x_n = B_{Saya}, \text{ di mana } B_{Saya} \geq 0 \\ &\text{tunduk pada} && \text{untuk } Saya = 1, 2, \ldots, mx_1 \geq 0, \ldots, x_n \geq 0. \end{aligned} \quad \textbf{(10.3)}$$

Itu juga dapat ditulis dalam notasi matriks kompak:

$$\begin{aligned} &\text{memaksimalkan} && CX \\ &\text{tunduk pada} && Kapak = B \\ & && x \geq 0, \end{aligned}$$

di mana

$$C = [C_1\ C_2 \ldots C_n], \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad A = \begin{bmatrix} A_{11} & A_{12} & \ldots & A_{1n} \\ \vdots & \vdots & & \vdots \\ A_{M1} & A_{M2} & \ldots & A_{MN} \end{bmatrix}, \quad B = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_M \end{bmatrix}.$$

Setiap linier masalah pemrograman dapat diubah menjadi setara masalah dalam bentuk standar. Jika fungsi tujuan perlu diminimalkan, itu dapat diganti dengan masalah yang setara dengan memaksimalkan fungsi tujuan yang sama dengan semua koefisiennya. C_J digantikan oleh C_J, $J = 1, 2, \ldots, n$ (lihat Bagian 6.6 untuk diskusi yang lebih umum tentang transformasi tersebut). Jika kendala diberikan sebagai pertidaksamaan, itu dapat diganti dengan persamaan yang setara dengan menambahkan a ***variabel kendur*** mewakili perbedaan antara dua sisi pertidaksamaan asli. Misalnya, dua pertidaksamaan masalah (10.2) dapat ditransformasikan, masing-masing, ke dalam persamaan berikut:

$$x + kamu + kamu = 4 \text{ dimana } kamu \geq 0 \text{ dan } x + 3kamu + v = 6 \text{ dimana } v \geq 0.$$

Akhirnya, dalam sebagian besar masalah pemrograman linier, variabel harus nonnegatif untuk memulai karena mereka mewakili beberapa kuantitas fisik. Jika hal ini tidak terjadi dalam pernyataan awal dari suatu masalah, variabel tidak dibatasi x_J dapat diganti dengan selisih antara dua variabel nonnegatif baru: $x_J = x_J' - x_J'', x_J' \geq 0, x_J'' \geq 0.$

Jadi, masalah (10.2) dalam bentuk standar adalah program linier berikut: masalah dalam empat variabel:

$$\begin{aligned} &\text{memaksimalkan} && 3x + 5kamu + 0kamu + 0v \\ &\text{tunduk pada} && x + kamu + kamu = 4 \\ & && x + 3kamu + v = 6 \\ & && x, y, u, v \geq 0. \end{aligned} \tag{10.4}$$

Sangat mudah untuk melihat bahwa jika kita menemukan solusi optimal $(x^*, kamu^*, kamu^*, v^*)$ untuk masalah (10.4), kita dapat memperoleh solusi optimal untuk masalah (10.2) hanya dengan mengabaikan dua koordinat terakhirnya.

Keuntungan utama dari bentuk standar terletak pada mekanisme sederhana yang disediakan untuk mengidentifikasi titik-titik ekstrim dari daerah yang layak. Untuk melakukan ini untuk

masalah (10.4), misalnya, kita perlu menetapkan dua dari empat variabel dalam persamaan kendala menjadi nol untuk mendapatkan sistem dua persamaan linier dalam dua yang tidak diketahui dan menyelesaikan sistem ini. Untuk kasus umum dari masalah dengan M persamaan dalam n tidak diketahui $(n \geq M)$, $n - M$ variabel perlu disetel ke nol untuk mendapatkan sistem M persamaan dalam M tidak diketahui. Jika sistem yang diperoleh memiliki solusi unik—seperti halnya sistem persamaan linier tak-degenerasi dengan jumlah persamaan yang sama dengan jumlah yang tidak diketahui—kita memiliki ***solusi dasar***; koordinatnya diatur ke nol sebelum menyelesaikan sistem disebut ***tidak dasar***, dan koordinatnya yang diperoleh dengan menyelesaikan sistem disebut ***dasar***. (Terminologi ini berasal dari aljabar linier.

Secara khusus, kita dapat menulis ulang sistem persamaan kendala (10.4) sebagai

$$x + kamu + kamu + v \begin{bmatrix} 1 & 1 & 1 \\ 1 & 3 & 0 & 1 \end{bmatrix} = \begin{matrix} 4 \\ 6 \end{matrix}.$$

Basis dalam ruang vektor dua dimensi terdiri dari dua vektor yang tidak proporsional satu sama lain; setelah basis dipilih, vektor apa pun dapat dinyatakan secara unik sebagai jumlah kelipatan dari vektor basis. Variabel dasar dan nonbasis menunjukkan vektor mana yang masing-masing disertakan dan dikecualikan dalam pilihan basis tertentu.)

Jika semua koordinat solusi basa nonnegatif, solusi dasar disebut a ***solusi layak dasar***. Misalnya, jika kita menetapkan variabel nol *x* dan *kamu* dan selesaikan sistem yang dihasilkan untuk *kamu* dan *v,* kita peroleh solusi fisibel dasar *(0, 0, 4, 6)* ; jika kita set ke nol variabel *x* dan *kamu* dan selesaikan sistem yang dihasilkan untuk *kamu* dan *v,* kita peroleh solusi dasarnya *(0, 4, 0, -6),* yang tidak layak. Pentingnya solusi layak dasar terletak pada korespondensi satu-ke-satu antara mereka dan titik-titik ekstrim dari wilayah layak. Sebagai contoh, *(0, 0, 4, 6)* adalah titik ekstrim dari daerah layak masalah (10.4) (dengan titik *(0, 0)* pada Gambar 10.1 menjadi proyeksinya pada *x, y* pesawat). Kebetulan, *(0, 0, 4, 6)* adalah titik awal alami untuk aplikasi metode simpleks untuk masalah ini.

Seperti disebutkan di atas, metode simpleks berkembang melalui serangkaian titik ekstrim yang berdekatan (solusi layak dasar) dengan peningkatan nilai fungsi tujuan. Setiap titik tersebut dapat diwakili oleh ***tablo simpleks***, sebuah tabel yang menyimpan informasi tentang solusi fisibel dasar yang sesuai dengan titik ekstrim. Misalnya, tablo simpleks untuk *(0, 0, 4, 6)* dari masalah (10.4) disajikan di bawah ini:

	x	*kamu*	*kamu*	*v*	
kamu	1	1	1	0	4
← *v*	1	3	0	1	6
	– 3	– 5	0	0	0
		↑			

(10.5)

Secara umum, tablo simpleks untuk masalah pemrograman linier dalam bentuk standar dengan *n* tidak diketahui dan *M* kendala persamaan linear *(n ≥ M)* memiliki *M* + 1 baris dan *n* + 1 kolom. Masing-masing yang pertama *M* baris tabel berisi koefisien persamaan kendala yang sesuai, dengan entri kolom terakhir berisi sisi kanan persamaan. Kolom, kecuali yang terakhir, diberi label dengan nama variabel. Baris diberi label oleh variabel dasar dari solusi layak dasar yang direpresentasikan tablo; nilai dari variabel dasar ini

solusi ada di kolom terakhir. Perhatikan juga bahwa kolom yang diberi label oleh variabel dasar membentuk$M \times M$ matriks identitas.

Baris terakhir dari tablo simpleks disebut ***baris objektif***. Ini diinisialisasi oleh koefisien fungsi tujuan dengan tanda-tanda mereka terbalik (dalam yang pertama$n$ kolom) dan nilai fungsi tujuan pada titik awal (pada kolom terakhir). Pada iterasi berikutnya, baris tujuan ditransformasikan dengan cara yang sama seperti semua baris lainnya. Baris objektif digunakan oleh metode simpleks untuk memeriksa apakah tablo saat ini mewakili solusi optimal: hal itu dilakukan jika semua entri dalam baris objektif—kecuali, mungkin, yang ada di kolom terakhir—nonnegatif. Jika tidak demikian, entri negatif mana pun menunjukkan variabel nonbasis yang dapat menjadi basis di tablo berikutnya.

Misalnya, menurut kriteria ini, solusi fisibel dasar $(0, 0, 4, 6)$ diwakili oleh tablo (10,5) tidak optimal. Nilai negatif dalam$x$-kolom menandakan fakta bahwa kita dapat meningkatkan nilai fungsi tujuan $z = 3x + 5kamu + 0kamu + 0v$ dengan meningkatkan nilai x-koordinat dalam solusi layak dasar saat ini $(0, 0, 4, 6)$. Memang, karena koefisien untuk$x$ dalam fungsi tujuan positif, semakin besar x nilai, semakin besar nilai fungsi ini. Tentu saja, kita perlu "mengkompensasi" peningkatan$x$ dengan menyesuaikan nilai variabel dasar $kamu$ dan v sehingga titik baru masih layak. Agar hal ini terjadi, kedua kondisi

$$x + kamu = 4 \quad \text{di mana } kamu \ge 0$$
$$x + v = 6 \quad \text{di mana } v \ge 0$$

harus puas, yang berarti bahwa

$$x \le \text{menit}\{4, 6\} = 4.$$

Perhatikan bahwa jika kita meningkatkan nilai x dari 0 hingga 4, jumlah terbesar yang mungkin, kita akan menemukan diri kita pada intinya $(4, 0, 0, 2)$, berdekatan dengan $(0, 0, 4, 6)$ titik ekstrim dari daerah layak, dengan $z = 12$. Demikian pula nilai negatif pada $kamu$

-kolom dari baris objektif menandakan

fakta bahwa kita juga dapat meningkatkan nilai fungsi tujuan dengan meningkatkan nilai $kamu$-koordinat dalam solusi layak dasar awal $(0, 0, 4, 6)$. Ini membutuhkan

$$kamu + kamu = 4 \quad \text{di mana } kamu \ge 0$$
$$3kamu + v = 6 \quad \text{di mana } v \ge 0,$$

yang berarti bahwa

$$kamu \le \text{menit}\{\frac{4}{1}, \frac{6}{3}\} = 2.$$

Jika kita meningkatkan nilai $kamu$ dari 0 hingga 2, jumlah terbesar yang mungkin, kita akan menemukan diri kita di titik $(0, 2, 2, 0)$, yang lain bersebelahan dengan $(0, 0, 4, 6)$ titik ekstrim, dengan $z = 10$.

Jika ada beberapa entri negatif dalam baris objektif, aturan yang umum digunakan adalah memilih yang paling negatif, yaitu, bilangan negatif dengan yang terbesar.

nilai mutlak. Aturan ini dimotivasi oleh pengamatan bahwa pilihan seperti itu menghasilkan peningkatan terbesar dalam nilai fungsi tujuan per unit perubahan dalam nilai variabel. (Dalam contoh kami, peningkatan x-nilai dari 0 sampai 1 at $(0, 0, 4, 6)$ mengubah nilai $z = 3x + 5kamu + 0kamu + 0v$ dari 0 menjadi 3, sementara peningkatan $kamu$-nilai dari 0 sampai 1 at $(0, 0, 4, 6)$ perubahan z dari 0 sampai 5.) Namun, perhatikan bahwa kendala kelayakan memberlakukan batasan yang berbeda pada seberapa banyak masing-masing variabel dapat meningkat. Dalam contoh kita, khususnya, pilihan $kamu$-variabel di atas x-variabel mengarah ke peningkatan yang lebih kecil dalam nilai fungsi tujuan. Namun, kami akan menggunakan aturan yang umum digunakan ini dan memilih variabel $kamu$ seperti yang kita lanjutkan dengan contoh kita. Sebuah variabel dasar baru disebut ***memasukkan variabel***, sedangkan kolomnya disebut sebagai ***kolom poros***; kami menandai kolom pivot dengan ↑ .Sekarang kami akan menjelaskan cara memilih

variabel keberangkatan, yaitu, variabel dasar menjadi nonbasic di tablo berikutnya. (Jumlah total variabel dasar dalam solusi dasar apa pun harus sama dengan M, jumlah kendala kesetaraan.) Seperti yang kita lihat di atas, untuk sampai ke titik ekstrim yang berdekatan dengan nilai yang lebih besar dari fungsi tujuan, kita perlu meningkatkan variabel masuk dengan jumlah terbesar untuk membuat salah satu variabel dasar lama nol sambil menjaga nonnegatif dari semua yang lain. Kita dapat menerjemahkan pengamatan ini ke dalam aturan berikut untuk memilih variabel keberangkatan dalam tabel simpleks: untuk masing-masing *positif* entri di kolom pivot, hitung θ-***perbandingan*** dengan membagi entri terakhir baris dengan entri di kolom pivot. Untuk contoh tablo (10.5), ini θ-rasio adalah

$$\theta_{kamu} = \frac{4}{1} = 4, \qquad \theta_v = \frac{6}{3} = 2.$$

Baris dengan yang terkecil θ-perbandingan menentukan variabel berangkat, yaitu, the variabel menjadi nonbasic. Ikatan dapat diputuskan secara sewenang-wenang. Untuk contoh kita, ini adalah variabel v. Kami menandai baris dari variabel yang berangkat, yang disebut ***baris pivot***, oleh ← -dan tunjukkan itu ← row. Perhatikan bahwa jika tidak ada entri positif di kolom pivot, tidak θ-rasio dapat dihitung, yang menunjukkan bahwa masalahnya tidak terbatas dan algoritma berhenti.

Akhirnya, langkah-langkah berikut perlu diambil untuk mengubah tablo saat ini menjadi yang berikutnya. (Transformasi ini, disebut ***berputar***, mirip dengan langkah utama dari algoritma eliminasi Gauss-Jordan untuk menyelesaikan sistem persamaan linier—lihat Soal 8 dalam Latihan 6.2.) Pertama, bagi semua entri dari baris pivot dengan ***poros*** , entrinya di kolom pivot, untuk mendapatkan row_{baru}. Untuk tablo (10.5), kita peroleh

$$\leftarrow \text{row}_{\text{baru}}: \quad \frac{1}{3} \quad 1 \; 0 \quad \frac{1}{3} \quad 2.$$

Kemudian, ganti setiap baris lainnya, termasuk baris objektif, dengan selisihnya

$$\text{baris } C \cdot \leftarrow \text{row}_{\text{baru}},$$

di mana C adalah entri baris di kolom pivot. Untuk tablo (10,5), ini menghasilkan

$$\text{baris 1 1} \ldots \leftarrow \text{row}_{\text{baru}}: \quad \frac{2}{3} \quad 0 \quad 1 \quad -\frac{1}{3} \quad 2,$$

$$\text{baris 3 (5)} \ldots \leftarrow \text{row}_{\text{baru}}: \quad -\frac{4}{3} \quad 0 \quad 0 \quad \frac{5}{3} \quad 10.$$

Dengan demikian, metode simpleks mengubah tablo (10,5) menjadi tablo berikut:

	x	*kamu*	*kamu*	*v*	
← *kamu*	$\frac{2}{3}$	0	1	$-\frac{1}{3}$	2
kamu	$\frac{1}{3}$	1	0	$\frac{1}{3}$	2
	$-\frac{4}{3}$	0	0	$\frac{5}{3}$	10
		↑			

(10.6)

Tableau (10.6) mewakili solusi layak dasar *(*0, 2, 2, 0*)* dengan peningkatan nilai fungsi tujuan, yaitu sebesar 10. Namun, itu tidak optimal (mengapa?).

Iterasi berikutnya—lakukan sendiri sebagai latihan yang bagus!—menghasilkan tablo (10.7):

	x	*kamu*	*kamu*	*v*	
x	1	0	$\frac{3}{2}$	$-\frac{1}{2}$	3
kamu	0	1	$-\frac{1}{2}$	$\frac{1}{2}$	1
	0	0	2	1	14

(10.7)

Tablo ini mewakili solusi dasar yang layak *(*3, 1, 0, 0*)*. Ini optimal karena semua entri pada baris objektif tablo (10.7) adalah nonnegatif. Nilai maksimal dari fungsi tujuan sama dengan 14, entri terakhir di baris tujuan.

Mari kita rangkum langkah-langkah metode simpleks.

Ringkasan metode simpleks

Langkah 0 *inisialisasi* Menyajikan masalah pemrograman linier yang diberikan secara bentuk dard dan buat tablo awal dengan entri nonnegatif di kolom paling kanan dan *M* kolom lain yang menyusun *M* × *M* matriks identitas. (Entri di baris tujuan harus diabaikan dalam memverifikasi persyaratan ini.) Ini *M* kolom mendefinisikan variabel dasar dari solusi layak dasar awal, yang digunakan sebagai label baris tablo.

Langkah 1 *Uji Optimalitas* Jika semua entri dalam baris tujuan (kecuali, mungkin, yang ada di kolom paling kanan, yang mewakili nilai dari

fungsi objektif) adalah nonnegatif—berhenti: tablo mewakili solusi optimal yang nilai variabel dasarnya berada di kolom paling kanan dan nilai variabel nonbasis lainnya adalah nol.

Langkah 2 *Menemukan variabel yang masuk* Pilih entri negatif dari antara pertama n elemen baris objektif. (Aturan yang umum digunakan adalah memilih entri negatif dengan nilai absolut terbesar, dengan ikatan putus secara sewenang-wenang.) Tandai kolomnya untuk menunjukkan variabel yang masuk dan kolom pivot.

Langkah 3 *Menemukan variabel berangkat* Untuk setiap entri positif di pivot kolom, hitung θ-rasio dengan membagi entri baris itu di kolom paling kanan dengan entrinya di kolom pivot. (Jika semua entri dalam kolom pivot negatif atau nol, masalahnya tidak terbatas—berhenti.) Temukan baris dengan yang terkecil θ-rasio (ikatan dapat diputus secara sewenang-wenang), dan tandai baris ini untuk menunjukkan variabel yang berangkat dan baris pivot.

Langkah 4 *Membentuk tablo berikutnya* Bagilah semua entri di baris pivot dengan entrinya di kolom pivot. Kurangi dari setiap baris lainnya, termasuk baris tujuan, baris pivot baru dikalikan dengan entri di kolom pivot dari baris yang bersangkutan. (Ini akan membuat semua entri di kolom pivot 0 kecuali 1 di baris pivot.) Ganti label baris pivot dengan nama variabel kolom pivot dan kembali ke Langkah 1.

Catatan Lebih Lanjut tentang Metode Simpleks

Bukti formal keabsahan langkah-langkah metode simpleks dapat ditemukan dalam buku-buku yang ditujukan untuk pembahasan rinci tentang pemrograman linier (misalnya, [Dan63]). Namun, beberapa catatan penting tentang metode ini masih perlu dibuat. Secara umum, sebuah iterasi dari metode simpleks mengarah ke titik ekstrim dari daerah fisibel masalah dengan nilai fungsi tujuan yang lebih besar. Dalam kasus degenerasi, yang muncul ketika satu atau lebih variabel dasar sama dengan nol, metode simpleks hanya dapat menjamin bahwa nilai fungsi tujuan pada titik ekstrem baru lebih besar atau sama dengan nilainya pada titik sebelumnya. Pada gilirannya, ini membuka pintu untuk kemungkinan tidak hanya bahwa nilai-nilai fungsi tujuan "menghentikan" untuk beberapa iterasi berturut-turut tetapi bahwa algoritme mungkin berputar kembali ke titik yang dipertimbangkan sebelumnya dan karenanya tidak pernah berhenti. Fenomena terakhir disebut ***bersepeda***. Meskipun jarang jika pernah terjadi dalam praktik, contoh spesifik masalah di mana bersepeda memang terjadi telah dibangun. Modifikasi sederhana dari Langkah 2 dan 3 dari metode simpleks, disebut ***Aturan Bland***, menghilangkan bahkan kemungkinan teoritis bersepeda. Dengan asumsi bahwa variabel dilambangkan dengan huruf subscript (misalnya, $x_1, x_2, \ldots, x_n$), aturan ini dapat dinyatakan sebagai berikut:

Langkah 2 dimodifikasi Di antara kolom dengan entri negatif di objektif baris, pilih kolom dengan subskrip terkecil.

Langkah 3 dimodifikasi Selesaikan dasi di antara yang terkecil θ-rasio dengan memilih baris yang diberi label oleh variabel dasar dengan subskrip terkecil.

Peringatan lain berkaitan dengan asumsi yang dibuat pada Langkah 0. Mereka secara otomatis terpenuhi jika masalah diberikan dalam bentuk di mana semua kendala dikenakan pada variabel nonnegatif adalah pertidaksamaan $A_{Saya1}x_1 + \ldots + A_{di\ dalam}x_n \le B_{Saya}$ dengan $B_{Saya} \ge 0$ untuk $Saya = 1, 2, \ldots, M$. Memang, dengan menambahkan variabel slack non-negatif x_{n+Saya} ke dalam $Saya$kendala th, kami memperoleh kesetaraan $A_{Saya1}x_1 + \ldots + A_{di\ dalam}x_n + x_{n+Saya} = B_{Saya}$, dan semua persyaratan yang dikenakan pada tablo awal metode simpleks dipenuhi untuk solusi layak dasar yang jelas $x_1 = \ldots = x_n = 0$, $x_{n+1} = \ldots = x_{n+M} = 1$. Tetapi jika masalah tidak diberikan dalam bentuk seperti itu, menemukan solusi layak dasar awal dapat menghadirkan hambatan nontrivial. Selain itu, untuk masalah dengan daerah layak yang kosong, tidak ada solusi layak dasar awal, dan kita memerlukan cara algoritmik untuk mengidentifikasi masalah tersebut. Salah satu cara untuk mengatasi masalah ini adalah dengan menggunakan ekstensi ke metode simpleks klasik yang disebut ***metode simpleks dua fase*** (lihat, misalnya, [Kol95]). Singkatnya, metode ini menambahkan satu set variabel buatan ke kendala kesetaraan dari masalah yang diberikan sehingga masalah baru memiliki solusi layak dasar yang jelas. Ini kemudian memecahkan masalah pemrograman linier meminimalkan jumlah variabel buatan dengan metode simpleks. Solusi optimal untuk masalah ini menghasilkan tablo awal untuk masalah asli atau menunjukkan bahwa daerah fisibel dari masalah asli kosong.

Seberapa efisienkah metode simpleks? Karena algoritma berkembang melalui urutan titik-titik yang berdekatan dari wilayah yang layak, kita mungkin harus mengharapkan kabar buruk karena jumlah titik ekstrim diketahui tumbuh secara eksponensial dengan ukuran masalah. Memang, efisiensi kasus terburuk dari metode simpleks telah terbukti eksponensial juga. Untungnya, lebih dari setengah abad pengalaman praktis dengan algoritme telah menunjukkan bahwa jumlah iterasi dalam aplikasi tipikal berkisar antara M dan $3M$, dengan jumlah operasi per iterasi sebanding dengan $M\,N$, di mana M dan n adalah jumlah kendala kesetaraan dan variabel, masing-masing.

Sejak penemuannya pada tahun 1947, metode simpleks telah menjadi subjek studi intensif oleh banyak peneliti. Beberapa dari mereka telah bekerja pada perbaikan algoritma asli dan rincian implementasi yang efisien. Sebagai hasil dari upaya ini, program yang menerapkan metode simpleks telah dipoles hingga masalah yang sangat besar dengan ratusan ribu kendala dan variabel dapat diselesaikan secara rutin. Bahkan, program tersebut telah berkembang menjadi paket perangkat lunak yang canggih. Paket-paket ini memungkinkan pengguna untuk memasukkan batasan masalah dan mendapatkan solusi dalam bentuk yang mudah digunakan. Mereka juga menyediakan alat untuk menyelidiki properti penting dari solusi, seperti kepekaannya terhadap perubahan data input. Investigasi semacam itu sangat penting untuk banyak aplikasi, termasuk di bidang ekonomi.

Para peneliti juga mencoba menemukan algoritma untuk memecahkan masalah pemrograman linier dengan efisiensi waktu polinomial dalam kasus terburuk. Tonggak penting dalam sejarah algoritme semacam itu adalah bukti oleh LG Khachian [Kha79] yang menunjukkan bahwa ***metode elips*** dapat menyelesaikan masalah pemrograman linier apa pun di

waktu polinomial. Meskipun metode ellipsoid jauh lebih lambat daripada metode simpleks dalam praktiknya, efisiensi terburuknya yang lebih baik mendorong pencarian alternatif untuk metode simpleks. Pada tahun 1984, Narendra Karmarkar menerbitkan sebuah algoritma yang tidak hanya memiliki efisiensi kasus terburuk polinomial tetapi juga kompetitif dengan metode simpleks dalam pengujian empiris. Meskipun kita tidak akan membahas***Algoritma Karmarkar*** [Kar84] di sini, perlu ditunjukkan bahwa ini juga didasarkan pada ide perbaikan berulang. Namun, algoritma Karmarkar menghasilkan urutan solusi layak yang terletak di dalam wilayah layak daripada melalui urutan titik ekstrim yang berdekatan seperti yang dilakukan metode simpleks. Algoritma semacam itu disebut***metode titik interior*** (lihat, misalnya, [Arb93]).

Latihan 10.1

1. Pertimbangkan versi berikut dari masalah lokasi kantor pos (Soal 3 dalam Latihan 3.3): Diberikan n bilangan bulat $x_1, x_2, \ldots, x_n$ mewakili koordinat dari n desa yang terletak di sepanjang jalan lurus, cari lokasi kantor pos yang meminimalkan jarak rata-rata antar desa. Kantor pos mungkin, tetapi tidak wajib, terletak di salah satu desa. Rancang algoritma perbaikan berulang untuk masalah ini. Apakah ini cara yang efisien untuk menyelesaikan masalah ini?

2. Selesaikan masalah program linier berikut secara geometris.

A.

$$\begin{aligned} \text{memaksimalkan} \quad & 3x + kamu \\ \text{tunduk pada} \quad & -x + kamu \le 1 \\ & 2x + kamu \le 4 \\ & x \ge 0,\ kamu \ge 0 \end{aligned}$$

B.

$$\begin{aligned} \text{memaksimalkan} \quad & x + 2kamu \\ \text{tunduk pada} \quad & 4x \ge kamu \\ & kamu \le 3 + x \\ & x \ge 0,\ kamu \ge 0 \end{aligned}$$

3. Pertimbangkan masalah pemrograman linier

$$\begin{aligned} \text{memperkecil} \quad & C_1 x + C_2 kamu \\ \text{tunduk pada} \quad & x + kamu \ge 4 \\ & x + 3kamu \ge 6 \\ & x \ge 0,\ kamu \ge 0 \end{aligned}$$

di mana C_1 dan C_2 adalah beberapa bilangan real yang keduanya tidak sama dengan nol.

A. Berikan contoh nilai koefisien C_1 dan C_2 dimana masalahnya memiliki solusi optimal yang unik.

B. Berikan contoh nilai koefisien C_1 dan C_2 yang masalahnya memiliki banyak solusi optimal yang tak terhingga.

C. Berikan contoh nilai koefisien C_1 dan C_2 yang masalahnya tidak memiliki solusi optimal.

4. Apakah solusi untuk masalah (10.2) akan berbeda jika kendala pertidaksamaannya ketat, yaitu, $x + y < 4$ dan $x + 3y < 6$, masing-masing?

5. Lacak metode simpleks pada

A. masalah Latihan 2a.

B. masalah Latihan 2b.

6. Telusuri metode simpleks pada masalah Contoh 1 di Bagian 6.6

A. dengan tangan.

B. dengan menggunakan salah satu implementasi yang tersedia di Internet.

7. Tentukan berapa banyak iterasi yang dibutuhkan metode simpleks untuk menyelesaikan masalah!

$$\begin{aligned} &\text{memaksimalkan} && \sum_{j=1}^{n} x_j \\ &\text{tunduk pada} && 0 \le x_j \le B_j, \text{ di mana } B_j > 0 \text{ untuk } j = 1, 2, \ldots, n. \end{aligned}$$

8. Bisakah kita menerapkan metode simpleks untuk menyelesaikan masalah knapsack (lihat Contoh 2 di Bagian 6.6)? Jika Anda menjawab ya, tunjukkan apakah itu algoritma yang baik untuk masalah yang bersangkutan; jika Anda menjawab tidak, jelaskan mengapa tidak.

9. Buktikan bahwa tidak ada solusi masalah dapat memiliki persis $k \ge 1$ optimal pemrograman linier kecuali $k = 1$.

10. Jika masalah program linier

$$\begin{aligned} &\text{memaksimalkan} && \sum_{j=1}^{n} C_j x_j \\ &\text{tunduk pada} && \sum_{j=1}^{n} A_{aku\,j} x_j \le B_{Saya} \text{ untuk } Saya = 1, 2, \ldots, M \\ & && x_1, x_2, \ldots, x_n \ge 0 \end{aligned}$$

dianggap sebagai ***utama***, maka ***ganda*** didefinisikan sebagai program linier masalah

$$\begin{aligned} &\text{memperkecil} && \sum_{Saya=1}^{M} B_{Saya} kamu_{Saya} \\ &\text{tunduk pada} && \sum_{Saya=1}^{M} A_{aku\,j}\, kamu_{Saya} \ge C_j \text{ untuk } j = 1, 2, \ldots, n \\ & && kamu_1, kamu_2, \ldots, kamu_M \ge 0. \end{aligned}$$

A. Nyatakan masalah primal dan dual dalam notasi matriks.

B. Temukan dual dari masalah pemrograman linier

$$\begin{aligned} \text{memaksimalkan} \quad & x_1 + 4x_2 - x_3 \\ \text{tunduk pada} \quad & x_1 + x_2 + x_3 \le 6 \\ & x_1 - x_2 2x_3 \le 2 \\ & x_1, x_2, x_3 \ge 0. \end{aligned}$$

C. Memecahkan masalah primal dan dual dan membandingkan nilai optimal dari fungsi tujuan mereka.

10.2 **Masalah Aliran Maksimum**

Pada bagian ini, kami mempertimbangkan masalah penting dalam memaksimalkan aliran material melalui jaringan transportasi (sistem perpipaan, sistem komunikasi, sistem distribusi listrik, dan sebagainya). Kami akan mengasumsikan bahwa jaringan transportasi yang bersangkutan dapat diwakili oleh digraf berbobot terhubung dengan *n* simpul bernomor dari 1 sampai *n* dan satu set tepi *E*, dengan sifat-sifat berikut:

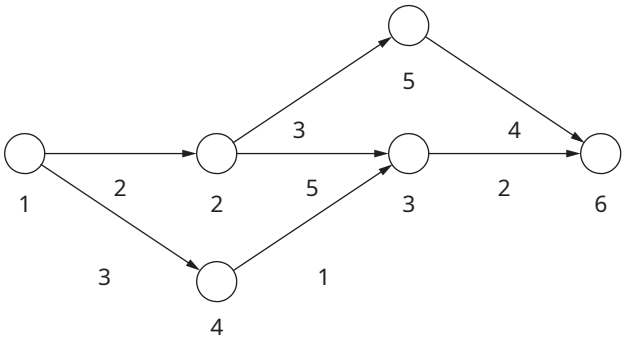
- Ini berisi tepat satu simpul tanpa tepi masuk; simpul ini disebut ***sumber*** dan dianggap bernomor 1.
- Ini berisi tepat satu simpul tanpa tepi yang meninggalkan; simpul ini disebut ***tenggelam*** dan diasumsikan diberi nomor *n*.
- beratnya $kamu_{aku\,j}$ dari setiap tepi terarah *(aku j)* adalah bilangan bulat positif, disebut tepian ***kapasitas***. (Angka ini mewakili batas atas jumlah materi yang dapat dikirim dari *Saya* ke *J* melalui tautan yang diwakili oleh tepi ini.)

Digraf yang memenuhi sifat-sifat ini disebut ***fljaringan ow*** atau hanya ***jaringan***.[3] Sebuah contoh kecil dari jaringan diberikan pada Gambar 10.4.

Diasumsikan bahwa sumber dan bak cuci adalah satu-satunya sumber dan tujuan material, masing-masing; semua simpul lainnya hanya dapat berfungsi sebagai titik di mana aliran dapat dialihkan tanpa mengkonsumsi atau menambahkan sejumlah materi. Dengan kata lain, jumlah total material yang memasuki titik tengah harus sama dengan jumlah total material yang keluar dari titik tersebut. Kondisi ini disebut ***fl persyaratan konservasi arus***. Jika kami menunjukkan jumlah yang dikirim melalui tepi *(aku j)* oleh $x_{aku\,j}$, maka untuk sembarang simpul perantara *Saya*, persyaratan aliran-konservasi dapat dinyatakan dengan kendala kesetaraan berikut:

$$\sum_{J:\,(Ji)\in E} x_{Ji} = \sum_{J:\,(aku\,j)\in E} x_{aku\,j} \text{ untuk } Saya = 2, 3, \ldots, n-1, \qquad \textbf{(10.8)}$$

3. Dalam model yang sedikit lebih umum, seseorang dapat mempertimbangkan jaringan dengan beberapa sumber dan sink dan memungkinkan kapasitas $kamu_{aku\,j}$ menjadi besar tak terhingga.



GAMBAR 10.4 Contoh grafik jaringan. Nomor simpul adalah "nama" simpul; nomor tepi adalah kapasitas tepi.

di mana jumlah di ruas kiri dan ruas kanan menyatakan total arus masuk dan keluar titik masuk dan keluar *Saya,* masing-masing.

Karena tidak ada jumlah material yang dapat berubah dengan melewati simpul perantara dari jaringan, dapat dipastikan bahwa jumlah total material yang meninggalkan sumber harus berakhir di wastafel. (Pengamatan ini juga dapat diturunkan secara formal dari persamaan (10.8), tugas yang akan Anda lakukan dalam latihan.) Jadi, kita memiliki persamaan berikut:

$$\sum_{j:(1,j)\in E} x_{1j} = \sum_{j:(j,n)\in E} x_{jn}. \qquad \textbf{(10.9)}$$

Kuantitas ini, total aliran keluar dari sumber—atau, secara ekuivalen, total aliran masuk ke bak cuci—disebut ***nilai*** dari aliran. Kami menyatakannya dengan *v*. Kuantitas inilah yang ingin kita maksimalkan pada semua aliran yang mungkin dalam jaringan.

Jadi, (layak) ***fladuh*** adalah penugasan bilangan real $x_{aku\,j}$ ke tepi *(aku j)* dari jaringan tertentu yang memenuhi kendala aliran-konservasi (10.8) dan ***keterbatasan kapasitas***

$$0 \le x_{aku\,j} \le kamu_{aku\,j} \text{ untuk setiap tepi } (aku\ j) \in E. \qquad \textbf{(10.10)}$$

NS ***masalah aliran maksimum*** dapat dinyatakan secara formal sebagai masalah optimasi berikut:

$$\text{memaksimalkan } v = \sum_{j:(1,j)\in E} x_{1j}$$

$$\text{tunduk pada } \sum_{j:(ji)\in E} x_{ji} - \sum_{j:(aku\,j)\in E} x_{aku\,j} = 0 \text{ untuk } Saya = 2, 3, \ldots, n-1 \qquad \textbf{(10.11)}$$

$$0 \le x_{aku\,j} \le kamu_{aku\,j} \quad \text{untuk setiap tepi } (aku\ j) \in E.$$

Kita dapat memecahkan masalah pemrograman linier (10.11) dengan metode simpleks atau oleh algoritma lain untuk masalah pemrograman linier umum (lihat Bagian 10.1).

Namun, struktur khusus masalah (10.11) dapat dimanfaatkan untuk merancang algoritma yang lebih cepat. Secara khusus, sangat wajar untuk menggunakan perbaikan berulang

ide sebagai berikut. Kita selalu dapat memulai dengan aliran nol (yaitu, set $x_{aku\ j} = 0$ untuk setiap tepian $(aku\ j)$ dalam jaringan). Kemudian, pada setiap iterasi, kita dapat mencoba menemukan jalur dari sumber ke sink di mana beberapa aliran tambahan dapat dikirim. Jalan seperti itu disebut ***flow menambah***. Jika jalur penambah aliran ditemukan, kami menyesuaikan aliran di sepanjang tepi jalur ini untuk mendapatkan aliran dengan nilai yang meningkat dan mencoba menemukan jalur penambah untuk aliran baru. Jika tidak ada jalur penambah aliran yang dapat ditemukan, kami menyimpulkan bahwa aliran arus optimal. Template umum untuk menyelesaikan masalah aliran maksimum ini disebut ***metode jalur tambahan*** , juga dikenal sebagai ***Metode Ford-Fulkerson*** setelah LR Ford, Jr., dan DR Fulkerson, yang menemukannya (lihat [Untuk57]).

Implementasi sebenarnya dari ide jalur augmentasi, bagaimanapun, tidak cukup mudah. Untuk melihat ini, mari kita perhatikan jaringan pada Gambar 10.4. Kita mulai dengan aliran nol yang ditunjukkan pada Gambar 10.5a. (Dalam gambar itu, jumlah nol yang dikirim melalui setiap tepi dipisahkan dari kapasitas tepi oleh garis miring; kami akan menggunakan notasi ini dalam contoh lain juga.) Adalah wajar untuk mencari jalur peningkatan aliran dari sumber ke tenggelam dengan mengikuti tepi terarah $(aku\ j)$ yang arusnya mengalir $x_{aku\ j}$ kurang dari kapasitas tepi $kamu_{aku\ j}$. Di antara beberapa kemungkinan, mari kita asumsikan bahwa kita mengidentifikasi jalur augmentasi 1→2→3→6 pertama. Kita dapat meningkatkan aliran sepanjang jalur ini dengan maksimal 2 unit, yang merupakan kapasitas terkecil yang tidak terpakai dari tepinya. Aliran baru ditunjukkan pada Gambar 10.5b. Sejauh ini gagasan sederhana kami tentang jalur peningkatan aliran akan dapat membawa kami. Sayangnya, aliran yang ditunjukkan pada Gambar 10.5b tidak optimal: nilainya masih dapat ditingkatkan sepanjang jalur 1→4→3←2→5→6 dengan meningkatkan aliran sebesar 1 di tepi $(1, 4)$, $(4, 3)$, $(2, 5)$, dan $(5, 6)$ dan *menurun* itu dengan 1 di tepi $(2, 3)$. Aliran yang diperoleh sebagai hasil dari augmentasi ini ditunjukkan pada Gambar 10.5c. Memang sudah maksimal. (Bisakah Anda memberi tahu mengapa?)

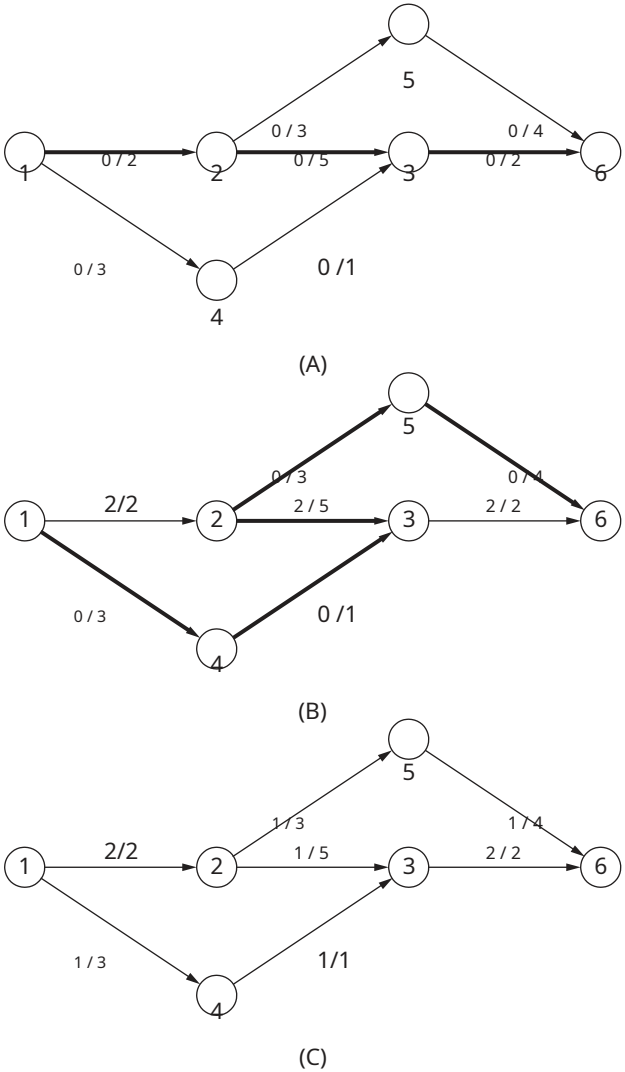
Jadi, untuk menemukan jalur penambah aliran untuk suatu aliran x, kita perlu mempertimbangkan jalur dari sumber untuk tenggelam di dasarnya *tidak terarah* grafik di mana setiap dua simpul berurutan $aku\ j$ baik?

Saya. dihubungkan oleh sisi berarah dari $Saya$ ke J dengan beberapa kapasitas positif yang tidak terpakai $R_{aku\ j} = kamu_{aku\ j} - x_{aku\ j}$ (sehingga kami dapat meningkatkan aliran melalui tepi itu hingga $R_{aku\ j}$ satuan), atau

ii. dihubungkan oleh sisi berarah dari J ke $Saya$ dengan beberapa aliran positif x_{Ji} (sehingga kita dapat mengurangi aliran melalui tepi itu hingga x_{Ji} unit).

Tepi jenis pertama disebut ***tepi depan*** karena ekor mereka terdaftar sebelum kepala mereka di daftar simpul $1 \rightarrow \ldots Saya \rightarrow J \ldots \rightarrow n$ menentukan jalan; tepi jenis kedua disebut ***tepi belakang*** karena ekor mereka terdaftar setelah kepala mereka di daftar jalur $1 \rightarrow \ldots Saya \leftarrow J \ldots \rightarrow n$. Sebagai ilustrasi, untuk jalur 1→4→3←2→5→6 contoh terakhir, $(1, 4)$, $(4, 3)$, $(2, 5)$, dan $(5, 6)$ adalah tepi depan, dan $(3, 2)$adalah tepi belakang.

Untuk jalur augmentasi aliran yang diberikan, mari R menjadi minimum dari semua kapasitas yang tidak terpakai $R_{aku\ j}$ dari tepi depan dan semua aliran x_{Ji} dari tepi belakangnya. Sangat mudah untuk melihat bahwa jika kita meningkatkan aliran arus sebesar R pada setiap tepi depan dan menguranginya dengan jumlah ini pada setiap tepi belakang, kita akan memperoleh nilai yang layak



GAMBAR 10.5 Ilustrasi metode augmenting-path. Jalur yang menambah aliran ditampilkan dalam huruf tebal. Jumlah aliran dan kapasitas tepi masing-masing ditunjukkan oleh angka sebelum dan sesudah garis miring.

aliran yang nilainya *R* unit lebih besar dari nilai pendahulunya. Memang, mari*Saya* menjadi titik tengah pada jalur yang menambah aliran. Ada empat kemungkinan kombinasi sisi maju dan sisi belakang yang datang ke simpul:*Saya*:

$$\underset{\rightarrow\ Saya\text{-}\rightarrow,}{+R\quad +R} \qquad \underset{\rightarrow\ Saya\leftarrow\text{-},}{+R\quad -R} \qquad \underset{\leftarrow\text{-}\ Saya\text{-}\rightarrow,}{-R\quad +R} \qquad \underset{\leftarrow\text{-}\ Saya\leftarrow\text{-}.}{-R\quad -R}$$

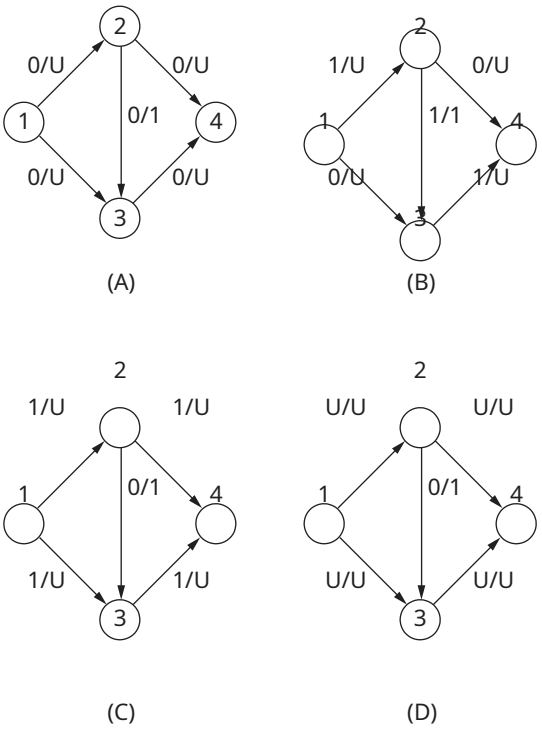
Untuk masing-masing dari mereka, persyaratan aliran-konservasi untuk vertex *Saya* akan tetap bertahan setelah penyesuaian aliran yang ditunjukkan di atas panah tepi. Selanjutnya, sejak*R* adalah minimum di antara semua kapasitas positif yang tidak terpakai di tepi depan dan semua aliran positif di tepi belakang jalur penambahan aliran, aliran baru akan memenuhi batasan kapasitas juga. Akhirnya, menambahkan*R* ke aliran di tepi pertama jalur augmentasi akan meningkatkan nilai aliran sebesar *R.*

Dengan asumsi bahwa semua kapasitas tepi adalah bilangan bulat, *R* akan menjadi bilangan bulat positif juga. Oleh karena itu, nilai aliran meningkat setidaknya 1 pada setiap iterasi dari metode jalur augmentasi. Karena nilai aliran maksimum dibatasi di atas (misalnya, dengan jumlah kapasitas tepi sumber), jalur augmenting metode harus berhenti setelah sejumlah iterasi terbatas.[4] Anehnya, aliran akhir selalu menjadi maksimal, terlepas dari urutan jalur tambahan. Hasil yang luar biasa ini berasal dari pembuktian Teorema Min-Cut Max-Aliran (lihat, misalnya, [For62]), yang kita replika nanti di bagian ini.

Metode augmenting-path—seperti yang dijelaskan di atas dalam bentuk umumnya—tidak menunjukkan cara khusus untuk menghasilkan jalur flow-augmenting. Urutan jalur yang buruk mungkin, bagaimanapun, memiliki dampak dramatis pada efisiensi metode. Perhatikan, misalnya, jaringan pada Gambar 10.6a, di mana:*kamu* singkatan dari beberapa bilangan bulat positif besar. Jika kita menambah aliran nol di sepanjang jalur 1→2→3→4, kita akan mendapatkan aliran nilai 1 yang ditunjukkan pada Gambar 10.6b. Menambah aliran itu di sepanjang jalur 1→3←2→4 akan meningkatkan nilai aliran menjadi 2 (Gambar 10.6c). Jika kita terus memilih pasangan jalur augmentasi aliran ini, kita akan membutuhkan total 2*kamu* iterasi untuk mencapai aliran maksimum nilai 2*kamu* (Gambar 10.6d). Tentu saja, kita dapat memperoleh aliran maksimum hanya dalam dua iterasi dengan menambah aliran nol awal sepanjang jalur 1→2→4 diikuti dengan menambah aliran baru di sepanjang jalur 1→3→4. Perbedaan dramatis antara 2*kamu* dan 2 iterasi membuat intinya.

Untungnya, ada beberapa cara untuk menghasilkan jalur augmentasi aliran secara efisien dan menghindari penurunan kinerja yang diilustrasikan oleh contoh sebelumnya. Yang paling sederhana dari mereka menggunakan pencarian luas-pertama untuk menghasilkan jalur augmentasi dengan jumlah tepi paling sedikit (lihat Bagian 3.5). Versi metode augmenting-path ini, disebut***terpendek-augmenting-path*** atau ***fialgoritme pemindaian pertama yang diberi label pertama***, disarankan oleh J. Edmonds dan RM Karp [Edm72]. Pelabelan mengacu pada penandaan simpul baru (tidak berlabel) dengan dua label. Label pertama menunjukkan jumlah aliran tambahan yang dapat dibawa dari sumber ke titik yang diberi label. Label kedua adalah nama simpul dari mana simpul yang diberi label tercapai. (Hal ini dapat dibiarkan tidak terdefinisi untuk sumbernya.) Juga mudah untuk menambahkan tanda + atau ke label kedua untuk menunjukkan apakah titik tersebut dicapai melalui tepi maju atau mundur, masing-masing. Sumber dapat selalu dilabeli dengan∞*, -.* Untuk simpul lainnya, label dihitung sebagai berikut.

4. Jika batas atas kapasitas adalah bilangan irasional, metode jalur augmentasi mungkin tidak berakhir (lihat, misalnya, [Chv83, hlm. 387–388], untuk contoh yang dirancang dengan cerdik yang menunjukkan situasi seperti itu). Batasan ini hanya menarik secara teoritis karena kita tidak dapat menyimpan bilangan irasional di komputer, dan bilangan rasional dapat diubah menjadi bilangan bulat dengan mengubah unit pengukuran kapasitas.



GAMBAR 10.6 Degradasi efisiensi metode augmenting-path.

Jika simpul tidak berlabel *J* terhubung ke simpul depan *Saya* dari antrian traversal oleh tepi terarah dari *Saya* ke *J* dengan kapasitas positif yang tidak terpakai $R_{aku\,j} = kamu_{aku\,j} - x_{aku\,j}$, kemudian puncak *J* diberi label dengan aku_J, $Saya^+$, di mana aku_J = menit{aku_{Saya}, $R_{aku\,j}$}.Jika simpul tidak berlabel *J* terhubung ke simpul depan *Saya* dari antrian traversal oleh tepi terarah dari *J* ke *Saya* dengan aliran positif x_{Ji}, lalu simpul *J* diberi label denganaku_J, $Saya^-$, di mana aku_J = menit{aku_{Saya}, x_{Ji}}.

Jika traversal yang ditingkatkan pelabelan ini berakhir dengan pelabelan wastafel, aliran arus dapat ditambah dengan jumlah yang ditunjukkan oleh label pertama wastafel. Augmentasi dilakukan di sepanjang jalur augmentasi yang dilacak dengan mengikuti label kedua vertex dari sink ke source: jumlah aliran saat ini meningkat di tepi depan dan menurun di tepi terbelakang dari jalur ini. Jika, di sisi lain, sink tetap tidak berlabel setelah antrian traversal menjadi kosong, algoritme mengembalikan aliran arus sebagai maksimum dan berhenti.

ALGORITMA *JalurAugmenting Terpendek(G)*

//Mengimplementasikan algoritma short-augmenting-path //
Input: Jaringan dengan sumber tunggal 1, sink tunggal *n*, dan //
kapasitas bilangan bulat positif $kamu_{aku\,j}$ di tepinya *(aku j)*

//Output: Aliran maksimum $x$menetapkan $x_{aku\,j}$ = 0 ke setiap tepi $(aku\,j)$ di jaringan, beri label sumbernya dengan ∞, - dan tambahkan sumber ke antrian kosong Q

```
sementara tidak Kosong(Q) melakukan
    Saya ← Depan (Q); Dequeue(Q)untuk
    setiap tepi dari Saya ke J melakukan      //tepi depan
        jika J tidak berlabel
            R_aku_j ← kamu_aku_j - x_aku_j
            jika R_aku_j > 0
                aku_J ← menit{aku_Saya , R    label J dengan aku_J , Saya+
                aku_j };Antrian(Q, j )
    untuk setiap tepi dari J ke Saya melakukan      //tepi mundur
        jika J tidak berlabel
            jika x_Ji > 0
                aku_J ← menit{aku_Saya        label J dengan aku_J , Saya-
                , x_Ji};Antrian(Q, j )
    jika wastafel telah diberi label
        //augment di sepanjang jalur augmentasi yang ditemukanJ ←n //mulai dari
        wastafel dan bergerak mundur menggunakan label keduaketika J = 1 //
        sumber belum tercapai
            jika label kedua dari vertex J adalah Saya+
                x_aku_j ← x_aku_j + aku_n
            lain //label kedua dari vertex J adalah Saya-
                x_Ji ← x_Ji - aku_n
            J ←Saya; Saya ← titik yang ditunjukkan oleh Sayalabel kedua
        hapus semua label simpul kecuali yang dari sumbernya
        diinisialisasi ulang Q dengan sumbernya
    kembali x //aliran arus maksimum
```

Aplikasi algoritma ini ke jaringan pada Gambar 10.4 diilustrasikan pada Gambar 10.7.

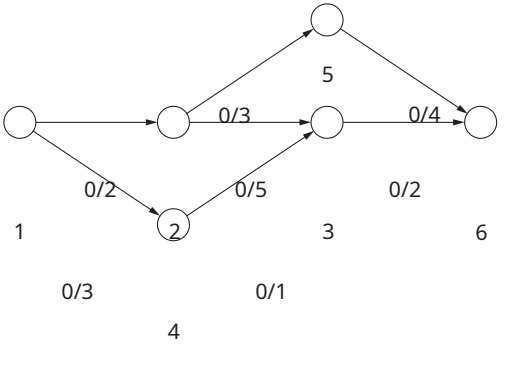
Optimalitas aliran akhir yang diperoleh dengan metode jalur augmentasi berasal dari teorema yang menghubungkan aliran jaringan dengan pemotongan jaringan. A ***memotong*** diinduksi dengan mempartisi simpul jaringan menjadi beberapa subset x berisi sumber dan X, pelengkap dari X, berisi wastafel adalah himpunan semua tepi dengan ekor di x dan kepala masuk X. Kami menunjukkan potongan $C(X, X)$ atau hanya C. Misalnya, untuk jaringan pada Gambar 10.4:

jika $x = \{1\}$ dan karenanya $X = \{2, 3, 4, 5, 6\}$, $C(X, X) = \{(1, 2), (1, 4)\}$;$C(X, X) =$

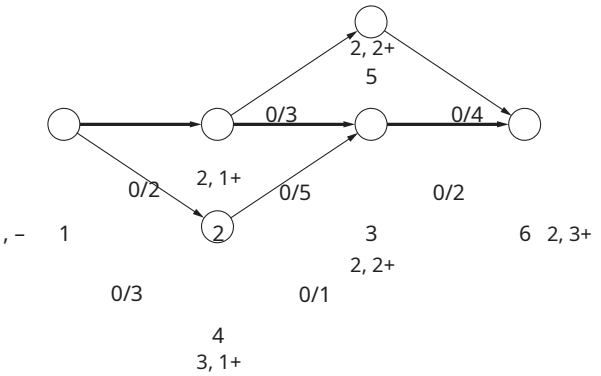
jika $x = \{1, 2, 3, 4, 5\}$ dan karenanya $X = \{6\}$, $\{(3, 6), (5, 6)\}$;$C(X, X) = \{(2, 3), (2$

jika $x = \{1, 2, 4\}$ dan karenanya $X = \{3, 5, 6\}$, $, 5), (4, 3)\}$.

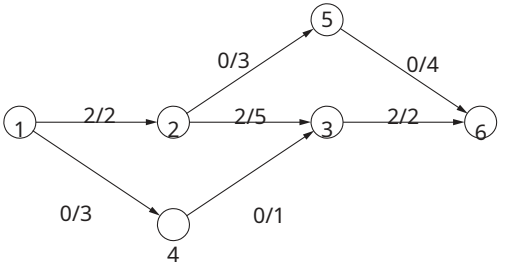
Nama "cut" berasal dari properti berikut: jika semua tepi potongan dihapus dari jaringan, tidak akan ada jalur terarah dari sumber ke tenggelam. Memang, mari $C(X, X)$ menjadi potongan. Pertimbangkan jalur terarah dari sumber ke tenggelam. Jika v_{Saya} adalah simpul pertama dari jalan yang dimiliki X (kumpulan simpul tersebut bukan



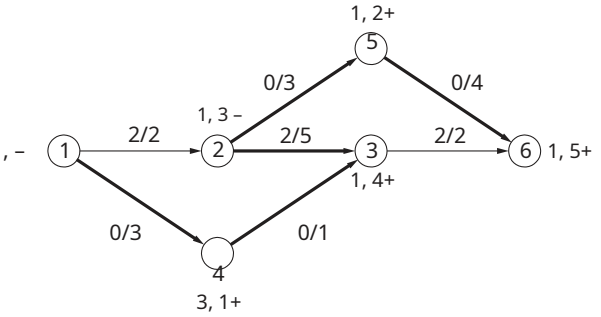
Antrian: 1 2 4 3 5 6
↑ ↑ ↑ ↑



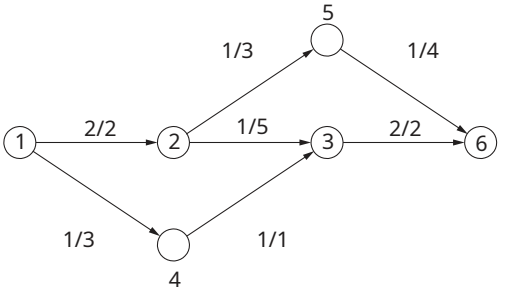
Tingkatkan aliran sebanyak 2 (label pertama wastafel)
sepanjang jalur 1 → 2 → 3 → 6.



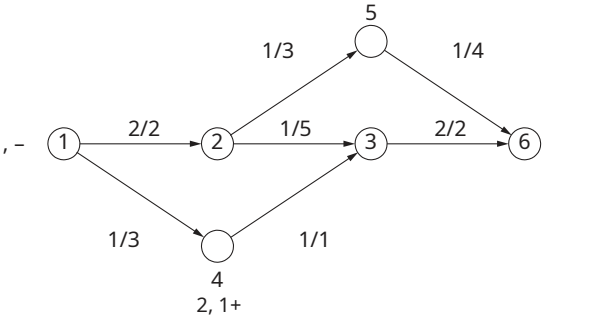
Antrian: 1 4 3 2 5 6
↑ ↑ ↑ ↑ ↑ ↑



Tingkatkan aliran sebanyak 1 (label pertama wastafel)
sepanjang jalur 1 → 4 → 3 2 → 5 → 6.



Antre: 1 4
↑ ↑



Tidak ada jalur tambahan (wastafel tidak berlabel);
arus yang mengalir maksimal.

GAMBAR 10.7 Ilustrasi algoritma jalur terpendek-augmenting. Diagram di sebelah kiri menunjukkan aliran saat ini sebelum iterasi berikutnya dimulai; diagram di sebelah kanan menunjukkan hasil pelabelan simpul pada iterasi itu, jalur augmentasi yang ditemukan (dicetak tebal), dan aliran sebelum augmentasinya. Simpul yang dihapus dari antrian ditunjukkan oleh↑simbol.

kosong, karena berisi wastafel), maka v_{Saya} bukan sumber dan pendahulu langsungnya v_{Saya1} di jalan itu milik X. Oleh karena itu, tepi dari v_{Saya1} ke v_{Saya} harus menjadi elemen potongan $C(X, X)$. Ini membuktikan properti yang dimaksud.

NS ***kapasitas*** dari potongan $C(X, X^-)$, dilambangkan $c(X, X^-)$, didefinisikan sebagai jumlah kapasitas tepi yang membentuk potongan. Untuk tiga contoh pemotongan yang diberikan di atas, kapasitasnya masing-masing sama dengan 5, 6, dan 9. Karena jumlah pemotongan yang berbeda dalam jaringan tidak kosong dan terbatas (mengapa?), selalu ada ***potongan minimum***, yaitu, pemotongan dengan kapasitas terkecil. (Berapakah potongan minimum dalam jaringan Gambar 10.4?) Teorema berikut menetapkan hubungan penting antara pengertian aliran maksimum dan potongan minimum.

TEOREMA (*Teorema Min-Cut Aliran Maks*) Nilai arus maksimum dalam suatu jaringan sama dengan kapasitas pemotongan minimumnya.

BUKTI Pertama, mari x menjadi aliran nilai yang layak v dan biarkan $C(X, X^-)$ menjadi potongan kapasitas C dalam jaringan yang sama. Pertimbangkan aliran melintasi potongan ini didefinisikan sebagai perbedaan antara jumlah aliran pada tepi dari x ke X dan jumlah aliran di tepi dari X ke x. Secara intuitif jelas dan dapat diturunkan secara formal dari persamaan yang menyatakan persyaratan konservasi aliran dan definisi nilai aliran (Soal 6b dalam latihan bagian ini) bahwa aliran melintasi potongan $C(X, X^-)$ adalah sama dengan v, nilai aliran:

$$v = \sum_{Saya \in X,\ j \in X} x_{aku\,j} - \sum_{j \in X^-,\ saya \in x} x_{ji}. \qquad \textbf{(10.12)}$$

Karena jumlah kedua adalah nonnegatif dan aliran $x_{aku\,j}$ di tepi manapun $(aku\,j)$ tidak dapat melebihi kapasitas tepi $kamu_{aku\,j}$, kesetaraan (10.12) menyiratkan bahwa

$$v \le \sum_{Saya \in X,\ j \in X} x_{aku\,j} \le \sum_{Saya \in X,\ j \in X} kamu_{aku\,j},$$

yaitu,

$$v \le C. \qquad \textbf{(10.13)}$$

Dengan demikian, nilai setiap aliran yang layak dalam suatu jaringan tidak dapat melebihi kapasitas setiap potongan dalam jaringan itu.

Membiarkan v^* menjadi nilai aliran akhir x^* diperoleh dengan metode augmenting-path. Jika sekarang kita menemukan potongan yang kapasitasnya sama dengan v^*, kita harus menyimpulkan, mengingat ketidaksetaraan (10.13), bahwa (i) nilai v^* aliran akhir maksimum di antara semua aliran yang layak, (ii) kapasitas pemotongan minimal di antara semua pemotongan dalam jaringan, dan (iii) nilai aliran maksimum sama dengan kapasitas potong minimum. Untuk menemukan potongan seperti itu, pertimbangkan himpunan simpul x^* yang dapat dicapai dari sumber dengan mengikuti jalur tidak berarah yang terdiri dari tepi depan dengan kapasitas positif yang tidak terpakai (berkenaan dengan aliran akhir x^*) dan tepi belakang dengan

arus positif pada mereka. Set ini berisi sumber tetapi tidak berisi sink: jika ya, kita akan memiliki jalur tambahan untuk aliran x^*, yang akan

bertentangan dengan asumsi bahwa aliran x^* adalah final. Pertimbangkan potongannya $C(X^*, \overline{X^*})$. Menurut definisi himpunan x^*, setiap tepi $(aku\ j)$ dari x^* ke $\overline{x^*}$ memiliki nol kapasitas yang tidak terpakai, yaitu, $x^*_{aku\ j} = kamu_{aku\ j}$, dan setiap tepi (Ji) dari x^* ke $\overline{x^*}$ memiliki aliran nol di atasnya (jika tidak, J akan masuk x^*). Menerapkan kesetaraan (10.12) ke aliran akhir x^* dan himpunan $\overline{x^*}$ didefinisikan di atas, kita peroleh

$$v^* = \sum_{Saya \in x^*,\, J \in \overline{x^*}} X^*_{aku\ j} - \sum_{J \in x^*,\, Saya \in \overline{x^*}} X^*_{Ji} = \sum_{Saya \in x^*,\, J \in \overline{x^*}} kamu_{aku\ j} - 0 = c(X^*, \overline{X^*}),$$

■

yang membuktikan teorema.

Pembuktian yang diuraikan di atas menyelesaikan lebih dari sekadar membuktikan kesetaraan nilai aliran maksimum dan kapasitas pemotongan minimum. Ini juga menyiratkan bahwa ketika metode jalur augmenting berakhir, itu menghasilkan aliran maksimum dan pemotongan minimum. Jika pelabelan dari jenis yang digunakan dalam algoritma jalur penambahan terpendek digunakan, potongan minimum dibentuk oleh tepi dari simpul berlabel ke tidak berlabel pada iterasi terakhir dari metode ini. Akhirnya, buktinya menyiratkan bahwa semua tepi tersebut harus penuh (yaitu, aliran harus sama dengan kapasitas tepi), dan semua tepi dari simpul yang tidak berlabel ke berlabel, jika ada, harus kosong (yaitu, memiliki nol arus pada mereka).). Secara khusus, untuk jaringan pada Gambar 10.7, algoritma menemukan potongan $\{(1, 2), (4, 3)\}$ kapasitas minimum 3, kedua tepinya penuh sesuai kebutuhan.

Edmonds dan Karp membuktikan dalam makalah mereka [Edm72] bahwa jumlah jalur augmenting yang dibutuhkan oleh algoritma shortest-augmenting-path tidak pernah melebihi $nm/2$, di mana n dan M adalah jumlah simpul dan tepi, masing-masing. Karena waktu yang diperlukan untuk menemukan jalur augmentasi terpendek dengan pencarian luas-pertama adalah $Pada + M) = O(m)$ untuk jaringan yang diwakili oleh daftar kedekatannya, efisiensi waktu dari algoritma jalur penambahan terpendek ada di $O(nm^2)$.

Algoritma yang lebih efisien untuk masalah aliran maksimum telah diketahui (lihat monografi [Ahu93], serta bab yang sesuai dalam buku seperti [Cor09] dan [Kle06]). Beberapa dari mereka menerapkan ide jalur tambahan dengan cara yang lebih efisien. Lainnya didasarkan pada konsep preflows. A ***aliran awal*** adalah aliran yang memenuhi batasan kapasitas tetapi tidak memenuhi persyaratan konservasi aliran. Setiap simpul diperbolehkan memiliki lebih banyak aliran yang memasuki simpul daripada meninggalkannya. Algoritme preflowpush memindahkan aliran berlebih ke sink sampai persyaratan konservasi aliran ditetapkan kembali untuk semua simpul perantara jaringan. Algoritme yang lebih cepat dari jenis ini memiliki efisiensi terburuk yang mendekati $O(nm)$. Perhatikan bahwa algoritma preflowpush berada di luar paradigma perbaikan berulang karena mereka tidak menghasilkan urutan solusi peningkatan yang memenuhi *semua* kendala masalah.

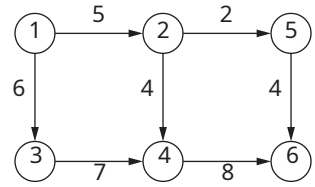
Untuk menyimpulkan bagian ini, perlu ditunjukkan bahwa meskipun minat awal dalam mempelajari aliran jaringan disebabkan oleh aplikasi transportasi, model ini juga terbukti berguna untuk banyak area lain. Kami membahas salah satunya di bagian berikutnya.

Latihan 10.2

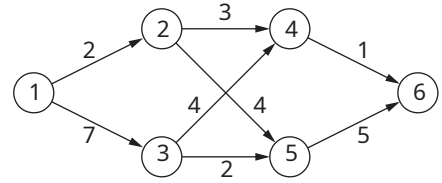
1. Karena algoritma aliran maksimum memerlukan tepi pemrosesan di kedua arah, akan lebih mudah untuk memodifikasi representasi matriks ketetanggaan dari jaringan sebagai berikut. Jika ada tepi berarah dari simpul *Saya* ke puncak *J* kapasitas $kamu_{aku\,j}$, maka elemen dalam *Saya* baris ke- *J* kolom ke $kamu_{aku\,j}$, dan elemen di *J* baris ke- *Saya* kolom th diatur ke $kamu_{aku\,j}$; jika tidak ada tepi antar simpul *Saya* dan *J*, kedua elemen ini disetel ke nol. Garis besar algoritma sederhana untuk mengidentifikasi sumber dan tenggelam dalam jaringan yang disajikan oleh matriks tersebut dan menunjukkan efisiensi waktunya.

2. Terapkan algoritma jalur penambahan terpendek untuk menemukan aliran maksimum dan pemotongan minimum pada jaringan berikut.

A.



B.



3. a. Apakah masalah aliran maksimum? selalu punya solusi unik? Akan jawaban Anda berbeda untuk jaringan dengan kapasitas berbeda di semua sisinya?

B. Jawab pertanyaan yang sama untuk masalah potongan minimum dalam menemukan potongan dengan kapasitas terkecil dalam jaringan tertentu.

4. a. Jelaskan bagaimana masalah aliran maksimum untuk jaringan dengan beberapa sumber dan sink dapat diubah menjadi masalah yang sama untuk jaringan dengan satu sumber dan satu sink.

B. Beberapa jaringan memiliki batasan kapasitas pada jumlah aliran yang dapat mengalir melalui simpul perantaranya. Jelaskan bagaimana masalah aliran maksimum untuk jaringan seperti itu dapat ditransformasikan ke masalah aliran maksimum untuk jaringan dengan batasan kapasitas tepi saja.

5. Pertimbangkan jaringan yang merupakan pohon berakar, dengan akar sebagai sumbernya, daun sebagai tenggelamnya, dan semua tepi diarahkan sepanjang jalur dari akar ke daun. Rancang algoritma yang efisien untuk menemukan aliran maksimum dalam jaringan seperti itu. Berapa efisiensi waktu algoritma Anda?

6. a. Buktikan kesetaraan (10.9).

B. Buktikan bahwa untuk setiap aliran dalam jaringan dan setiap potongan di dalamnya, nilai aliran sama dengan aliran melintasi potongan (lihat persamaan (10.12)). Jelaskan hubungan antara sifat ini dan persamaan (10.9).

7. a. Nyatakan masalah aliran maksimum untuk jaringan pada Gambar 10.4 sebagai masalah pemrograman linier.

B. Selesaikan masalah pemrograman linier ini dengan metode simpleks.

8. Sebagai alternatif dari algoritma shortest-augmenting-path, Edmonds dan Karp [Edm72] menyarankan algoritma maximum-capacity-augmenting-path, di mana aliran ditambah sepanjang jalur yang meningkatkan aliran dengan jumlah terbesar. Terapkan kedua algoritme ini dalam bahasa pilihan Anda dan lakukan penyelidikan empiris terhadap efisiensi relatifnya.

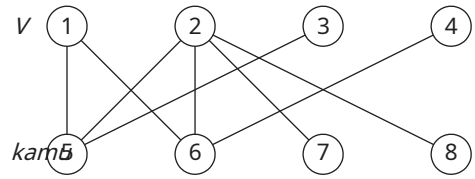
9. Tulis laporan tentang algoritme aliran maksimum yang lebih canggih seperti (i) algoritma Dinitz, (ii) algoritma Karzanov, (iii) algoritma Malhotra-Kamar-Maheshwari, atau (iv) algoritma Goldberg-Tarjan.

10. *Masalah makan* Beberapa keluarga pergi makan malam bersama. Untuk meningkatkan mereka interaksi sosial, mereka ingin duduk di meja sehingga tidak ada dua anggota keluarga yang sama berada di meja yang sama. Tunjukkan bagaimana menemukan pengaturan tempat duduk yang memenuhi tujuan ini (atau buktikan bahwa tidak ada pengaturan seperti itu) dengan menggunakan masalah aliran maksimum. Asumsikan bahwa kontingen makan malam memiliki P keluarga dan bahwa $Saya$ keluarga memiliki A_{Saya} anggota. Juga berasumsi bahwa Q meja tersedia dan J meja ini memiliki kapasitas tempat duduk B_J. [Ahu93]

10.3 Pencocokan Maksimum dalam Grafik Bipartit

Dalam banyak situasi kita dihadapkan pada masalah memasangkan elemen dari dua himpunan. Contoh tradisional adalah anak laki-laki dan perempuan untuk tarian, tetapi Anda dapat dengan mudah memikirkan aplikasi yang lebih serius. Lebih mudah untuk mewakili elemen dari dua himpunan yang diberikan oleh simpul dari grafik, dengan tepi di antara simpul yang dapat dipasangkan. A ***cocok*** dalam graf adalah himpunan bagian dari sisi-sisinya dengan sifat bahwa tidak ada dua sisi yang berbagi titik. A ***pencocokan maksimum***—lebih tepatnya, a ***pencocokan kardinalitas maksimum***—adalah pencocokan dengan jumlah tepi terbesar. (Apa itu untuk grafik pada Gambar 10.8? Apakah itu unik?) Masalah pencocokan maksimum adalah masalah menemukan pencocokan maksimum dalam grafik yang diberikan. Untuk grafik arbitrer, ini adalah masalah yang agak sulit. Itu diselesaikan pada tahun 1965 oleh Jack Edmonds [Edm65]. (Lihat [Gal86] untuk survei yang bagus dan referensi yang lebih baru.)

Kami membatasi diskusi kami di bagian ini pada kasus graf bipartit yang lebih sederhana. Di sebuah ***grafik bipartit***, semua simpul dapat dipartisi menjadi dua himpunan lepas V dan *kamu*, tidak harus berukuran sama, sehingga setiap sisi menghubungkan sebuah simpul di salah satu himpunan ini ke simpul di himpunan lainnya. Dengan kata lain, suatu graf adalah bipartit jika simpulnya dapat diwarnai dalam dua warna sehingga setiap sisi memiliki simpulnya yang diwarnai dengan warna yang berbeda; grafik seperti itu juga dikatakan ***2-warna***. Grafik pada Gambar 10.8 adalah bipartit. Tidak sulit untuk membuktikan bahwa suatu graf bipartit jika dan hanya jika tidak memiliki siklus dengan panjang ganjil. Kami akan berasumsi untuk sisa bagian ini bahwa



GAMBAR 10.8 Contoh graf bipartit.

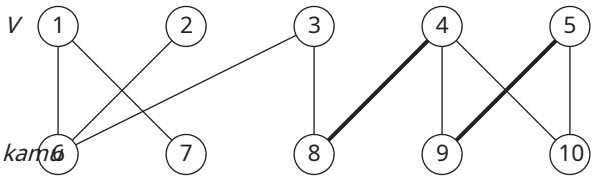
himpunan simpul dari graf bipartit yang diberikan telah dipartisi menjadi himpunan *V* dan *kamu* seperti yang dipersyaratkan oleh definisi (lihat Soal 8 dalam Latihan 3.5).

Mari kita terapkan teknik perbaikan iteratif pada masalah pencocokan kardinalitas maksimum. Membiarkan *M* menjadi kecocokan dalam graf bipartit *G* = *V, U, E.* Bagaimana kita bisa memperbaikinya, yaitu menemukan kecocokan baru dengan lebih banyak tepi? Jelas, jika setiap simpul di salah satu dari *V* atau *kamu* adalah ***cocok*** (mempunyai sebuah ***pasangan***), yaitu, berfungsi sebagai titik akhir dari tepi di *M*, ini tidak bisa dilakukan dan *M* adalah pencocokan maksimum. Oleh karena itu, untuk memiliki kesempatan meningkatkan pencocokan saat ini, keduanya *V* dan *kamu* harus mengandung ***tiada bandingan*** (disebut juga ***Gratis***) ***sudut***, yaitu, simpul-simpul yang tidak bersinggungan dengan sisi mana pun di *M.* Misalnya, untuk pencocokan M_A = {(4, 8), (5, 9)} dalam grafik pada Gambar 10.9a, simpul 1, 2, 3, 6, 7, dan 10 bebas, dan simpul 4, 5, 8, dan 9 cocok.

Pengamatan lain yang jelas adalah bahwa kita dapat segera meningkatkan pencocokan arus dengan menambahkan tepi antara dua simpul bebas. Misalnya, menambahkan (1, 6) ke pencocokan M_A = {(4, 8), (5, 9)} dalam grafik pada Gambar 10.9a menghasilkan pencocokan yang lebih besar M_B = {(1, 6), (4, 8), (5, 9)} (Gambar 10.9b). Sekarang mari kita coba mencari kecocokan yang lebih besar dari M_B dengan mencocokkan simpul 2. Satu-satunya cara untuk melakukan ini adalah dengan memasukkan edge (2, 6) dalam pencocokan baru. Penyertaan ini membutuhkan penghapusan (1, 6), yang dapat dikompensasikan dengan memasukkan (1, 7) dalam pencocokan baru. Pencocokan baru ini M_C = {(1, 7), (2, 6), (4, 8), (5, 9)} ditunjukkan pada Gambar 10.9c.

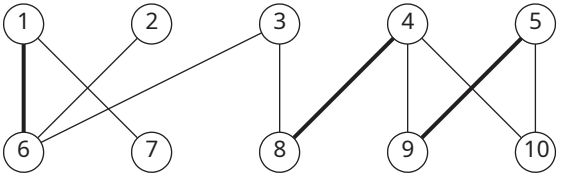
Secara umum, kami meningkatkan ukuran pencocokan saat ini *M* dengan membangun jalur sederhana dari titik bebas di *V* ke simpul bebas di *kamu* yang ujung-ujungnya berseberangan dalam *E* - *M* dan masuk *M.* Artinya, tepi jalan pertama bukan milik *M,* yang kedua melakukannya, dan seterusnya, sampai tepi terakhir yang bukan milik *M.* Jalan seperti itu disebut ***menambah*** sehubungan dengan pencocokan *M.* Misalnya, jalur 2, 6, 1, 7 adalah jalur augmentasi sehubungan dengan pencocokan M_B pada Gambar 10.9b.

Karena panjang jalur augmentasi selalu ganjil, menambah kecocokan *M* tepi jalur di posisi bernomor ganjil dan menghapus darinya tepi jalur di posisi bernomor genap menghasilkan pencocokan dengan satu tepi lebih banyak daripada di *M.* Penyesuaian yang cocok seperti itu disebut ***augmentasi***. Jadi, pada Gambar 10.9, pencocokan M_B diperoleh dengan augmentasi pencocokan M_A sepanjang menambah jalur 1, 6, dan pencocokan M_C diperoleh dengan augmentasi pencocokan M_B di sepanjang jalur augmentasi 2, 6, 1, 7. Bergerak lebih jauh, 3, 8, 4, 9, 5, 10 adalah jalur tambahan untuk pencocokan M_C (Gambar 10.9c). Setelah ditambahkan ke M_C ujung-ujungnya (3, 8), (4, 9), dan (5, 10) dan menghapus (4, 8) dan (5, 9), kami mendapatkan kecocokan M_D = {(1, 7), (2, 6), (3, 8), (4, 9), (5, 10)} ditunjukkan pada Gambar 10.9d. NS



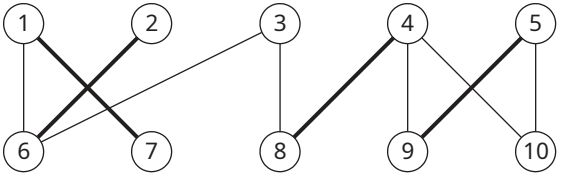
(A)

Jalur tambahan: 1, 6



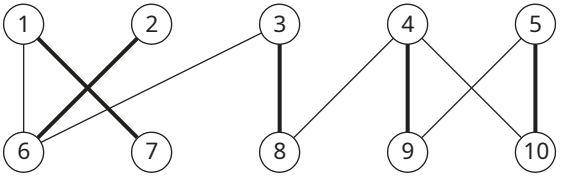
(B)

Jalur tambahan: 2, 6, 1, 7



(C)

Jalur tambahan: 3, 8, 4, 9, 5, 10



(D)

Pencocokan maksimum

GAMBAR 10.9 Menambah jalur dan mencocokkan augmentasi.

cocok M_D tidak hanya pencocokan maksimum tetapi juga ***sempurna*** , yaitu, pencocokan yang cocok dengan semua simpul dari grafik.

Sebelum kita membahas algoritme untuk menemukan jalur tambahan, mari kita selesaikan masalah apa arti ketiadaan jalur tersebut. Menurut teorema yang ditemukan oleh matematikawan Prancis Claude Berge, itu berarti pencocokan saat ini maksimal.

DALIL Sebuah pencocokan M adalah pencocokan maksimum jika dan hanya jika tidak ada jalur augmentasi terhadap M.

BUKTI Jika jalur augmentasi sehubungan dengan kecocokan M ada, maka ukuran pencocokan dapat ditingkatkan dengan augmentasi. Mari kita buktikan bagian yang lebih sulit: jika tidak ada jalur tambahan sehubungan dengan kecocokan M ada, maka pencocokan adalah pencocokan maksimum. Asumsikan bahwa, sebaliknya, ini bukan kasus untuk pencocokan tertentu M dalam grafik G. Membiarkan M^* menjadi pencocokan maksimum di G; dengan asumsi kami, jumlah tepi di M^* setidaknya satu lebih banyak dari jumlah sisi dalam M, yaitu $|M^*| > |M|$. Pertimbangkan tepi dalam perbedaan simetris $M \oplus M^* = (M - M^*) \cup (M^* - M)$, himpunan semua rusuk yang ada di M atau di M^* tapi tidak di keduanya. Perhatikan bahwa $|M^* - M| > |M - M^*|$ karena $|M^*| > |M|$ dengan asumsi. Membiarkan G menjadi subgraf dari G terdiri dari semua tepi di $M \oplus M^*$ dan titik akhir mereka. Menurut definisi pencocokan, setiap simpul di $G \subseteq G$ dapat insiden ke tidak lebih dari satu sisi di M dan tidak lebih dari satu sisi dalam M^*. Oleh karena itu, masing-masing simpul di G memiliki derajat 2 atau kurang, dan oleh karena itu setiap komponen terhubung dari G adalah jalur atau siklus panjang genap dari sisi bolak-balik dari $M - M^*$ dan $M^* - M$. Sejak $|M^* - M| > |M - M^*|$ dan jumlah rusuk dari $M - M^*$ dan $M^* - M$ adalah sama untuk setiap siklus panjang genap dari sisi bolak-balik di G , harus ada setidaknya satu jalur sisi bolak-balik yang dimulai dan diakhiri dengan sisi dari $M^* - M$. Oleh karena itu, ini adalah jalur tambahan untuk pencocokan M, yang bertentangan dengan asumsi bahwa tidak ada jalur seperti itu.

■

Diskusi kami tentang penambahan jalur mengarah ke metode umum berikut untuk membangun pencocokan maksimum dalam grafik bipartit. Mulailah dengan beberapa pencocokan awal (misalnya, himpunan kosong). Temukan jalur augmentasi dan tambahkan kecocokan saat ini di sepanjang jalur ini. Ketika tidak ada jalur augmenting yang dapat ditemukan, hentikan algoritma dan kembalikan pencocokan terakhir, yang maksimum.

Kami sekarang memberikan algoritma khusus yang mengimplementasikan template umum ini. Kami akan mencari jalur tambahan untuk pencocokan M oleh traversal seperti BFS dari grafik yang dimulai secara bersamaan di semua simpul bebas di salah satu himpunan V dan *kamu,* mengatakan, V . (Adalah logis untuk memilih yang lebih kecil dari dua himpunan simpul, tetapi kita akan mengabaikan pengamatan ini dalam kodesemu di bawah ini.) Ingat bahwa jalur tambahan, jika ada, adalah jalur panjang ganjil yang menghubungkan simpul bebas di V dengan simpul bebas di *kamu* dan yang, kecuali terdiri dari satu sisi, “zig” dari sebuah titik di V ke pasangan simpul lain di *kamu* , lalu "zags" kembali ke V sepanjang tepi yang didefinisikan secara unik dari M, dan seterusnya sampai titik bebas di *kamu* tercapai. (Gambarlah jalur augmenting untuk pencocokan pada Gambar 10.9, misalnya.) Oleh karena itu, setiap kandidat untuk menjadi seperti itu

jalur harus memiliki ujung-ujungnya bergantian dalam pola yang baru saja dijelaskan. Ini memotivasi aturan berikut untuk pelabelan simpul selama traversal seperti BFS dari grafik.

Kasus 1 (vertex depan antrian *w* ada di dalam *V)* Jika *kamu* adalah simpul bebas yang berdekatan dengan *w*, digunakan sebagai titik akhir lain dari jalur augmentasi; sehingga pelabelan berhenti dan augmentasi pencocokan dimulai. Jalur augmentasi yang dimaksud diperoleh dengan bergerak mundur sepanjang label simpul (lihat di bawah) untuk secara bergantian menambah dan menghapus tepinya ke dan dari pencocokan saat ini. Jika *kamu* tidak gratis dan terhubung ke *w* dengan tepi tidak di *M*, label *kamu* dengan *w* kecuali sudah diberi label.

Kasus 2 (puncak depan *w* ada di dalam *U)* Pada kasus ini, *w* harus cocok dan kita label pasangannya di *V* dengan *w.*

Berikut adalah pseudocode dari algoritma secara keseluruhan.

ALGORITMA *Pencocokan Bipartit Maksimum (G)*

//Menemukan kecocokan maksimum dalam grafik bipartit dengan traversal mirip BFS //Input: Grafik bipartit *G = V, U, E*//Output: Pencocokan kardinalitas maksimum *M* dalam grafik input, inisialisasi set *M* tepi dengan beberapa pencocokan yang valid (misalnya, himpunan kosong) menginisialisasi antrian *Q* dengan semua simpul gratis di *V* (dalam urutan apapun)**sementara tidak** *Kosong(Q)* **melakukan**

w ← *Depan (Q)*; *Dequeue(Q)***jika** *w* ∈ *V*

untuk setiap simpul *kamu* bersebelahan dengan *w* **melakukan**

jika *kamu* gratis

//menambah

M ← *M* ∪ *(Wu)*

v ← *w*

ketika *v* diberi label **melakukan**

kamu ← simpul yang ditunjukkan oleh *v* *M* ← *M* - *(v, kamu)*

labelnya; *v* ← simpul yang ditunjukkan oleh *M* ← *M* ∪ *(v, kamu)*

kamu labelnya; hapus semua label simpul

inisialisasi ulang *Q* dengan semua simpul gratis di *V*

merusak //keluar dari loop for

lain //*kamu* cocok

jika *(Wu)* ∈ *M* **dan** *kamu* tidak berlabel

label *kamu* dengan *w*

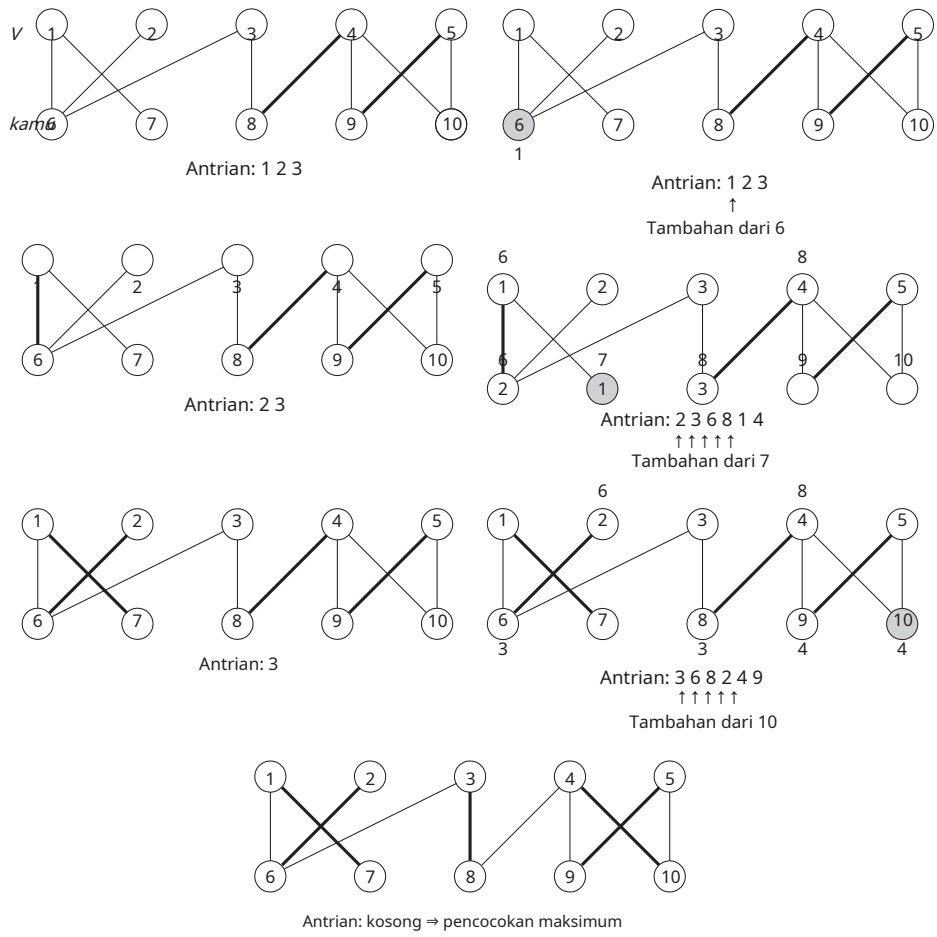
Antrian (Q, u)

lain //*w* ∈ *kamu* (dan cocok)

label jodoh *v* dari *w* dengan *w Antrian(Q, v)*

kembali *M* //pencocokan saat ini maksimum

Aplikasi dari algoritma ini untuk pencocokan pada Gambar 10.9a ditunjukkan pada Gambar 10.10. Perhatikan bahwa algoritme menemukan kecocokan maksimum yang berbeda dari yang ada di Gambar 10.9d.



GAMBAR 10.10 Penerapan algoritma pencocokan kardinalitas maksimum. Kolom kiri menunjukkan antrian yang cocok dan diinisialisasi saat ini pada awal iterasi berikutnya; kolom kanan menunjukkan pelabelan simpul yang dihasilkan oleh algoritma sebelum augmentasi dilakukan. Tepi yang cocok ditampilkan dalam huruf tebal. Label simpul menunjukkan simpul dari mana pelabelan dilakukan. Titik akhir yang ditemukan dari jalur augmentasi diarsir dan diberi label untuk kejelasan. Simpul yang dihapus dari antrian ditunjukkan oleh.

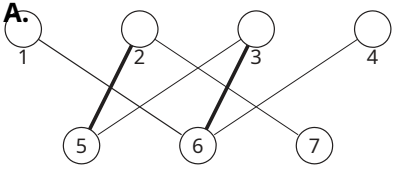
Seberapa efisien algoritma pencocokan maksimum? Setiap iterasi kecuali yang terakhir cocok dengan dua simpul bebas sebelumnya — satu dari masing-masing set *V* dan *U.* Oleh karena itu, jumlah total iterasi tidak boleh melebihi *n*/2 + 1, di mana *n* = | *V* | + | *kamu* | adalah jumlah simpul dalam grafik. Waktu yang dihabiskan untuk setiap iterasi adalah *Pada* + *M),* di mana *M* = | *E*| adalah jumlah sisi pada graf tersebut. (Ini mengasumsikan bahwa informasi tentang status setiap simpul — bebas atau cocok dan pasangan simpul jika yang terakhir — dapat diambil dalam waktu yang konstan, misalnya dengan menyimpannya dalam array.) Oleh karena itu, efisiensi waktu dari algoritma ada di dalam *O(n(n)+ M)*. Hopcroft dan Karp [Hop73] menunjukkan bagaimana efisiensi dapat ditingkatkan menjadi *O(n(n* + *M))* dengan menggabungkan beberapa iterasi ke dalam satu tahap untuk memaksimalkan jumlah tepi yang ditambahkan ke pencocokan dengan satu pencarian.

Kami prihatin di bagian ini dengan mencocokkan jumlah pasangan simpul terbesar yang mungkin dalam grafik bipartit. Beberapa aplikasi mungkin perlu mempertimbangkan kualitas atau biaya pencocokan pasangan yang berbeda. Misalnya, pekerja dapat melakukan pekerjaan dengan efisiensi yang berbeda, atau anak perempuan mungkin memiliki preferensi yang berbeda untuk calon pasangan dansa mereka. Wajar untuk memodelkan situasi seperti itu dengan grafik bipartit dengan bobot yang ditetapkan pada tepinya. Hal ini menyebabkan masalah memaksimalkan jumlah bobot pada tepi yang menghubungkan pasangan simpul yang cocok. Masalah ini disebut ***pencocokan berat maksimum***. Kami menemukannya dengan nama yang berbeda—masalah penugasan—di Bagian 3.4. Ada beberapa algoritma canggih untuk masalah ini, yang jauh lebih efisien daripada pencarian lengkap (lihat, misalnya, [Pap82], [Gal86], [Ahu93]). Namun, kita harus meninggalkannya di luar diskusi kita, karena kompleksitasnya, terutama untuk grafik umum.

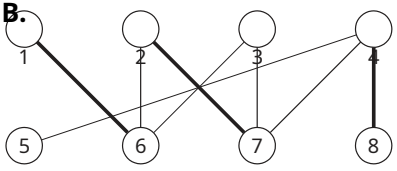
Latihan 10.3

1. Untuk setiap pencocokan yang ditunjukkan di bawah dalam huruf tebal, temukan augmentasi atau jelaskan mengapa tidak ada augmentasi.

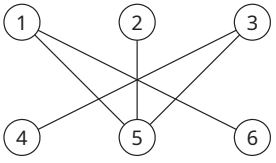
A.



B.



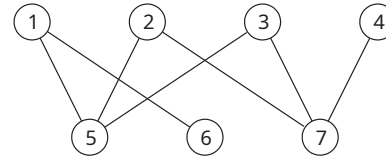
2. Terapkan algoritma pencocokan maksimum untuk grafik bipartit berikut:



3. a. Berapa kardinalitas terbesar dan terkecil yang mungkin dari suatu kecocokan pada graf bipartit? $G = V, U, E$ dengan n simpul di setiap himpunan simpul V dan *kamu* dan setidaknya n tepi?

B. Berapakah jumlah terbesar dan terkecil dari solusi yang berbeda yang dapat dimiliki oleh masalah pencocokan kardinalitas maksimum untuk graf bipartit $G = V, U, E$ dengan n simpul di setiap himpunan simpul V dan *kamu* dan setidaknya n tepi?

4. a. ***Teorema Pernikahan Hall*** menyatakan bahwa graf bipartit $G = V, U, E$ memiliki kecocokan yang cocok dengan semua simpul dari himpunan V jika dan hanya jika untuk setiap himpunan bagian $S \subseteq V$, $|R(S)| \geq |S|$ di mana $R(S)$ adalah himpunan semua simpul yang berdekatan dengan simpul di S. Periksa properti ini untuk grafik berikut dengan (i) $V = \{1, 2, 3, 4\}$ dan (ii) $V = \{5, 6, 7\}$.



B. Anda harus merancang algoritma yang mengembalikan ya jika ada kecocokan dalam grafik bipartit $G = V, U, E$ yang cocok dengan semua simpul di V dan tidak mengembalikan sebaliknya. Apakah Anda akan mendasarkan algoritme Anda pada pemeriksaan kondisi Teorema Perkawinan Hall?

5. Misalkan ada lima komite $A, B, C, D,$ dan E terdiri dari enam orang $a, b, c, d, e,$ dan F sebagai berikut: panitia A anggotanya adalah B dan e; komite B anggotanya adalah B, D, dan e; komite C anggotanya adalah $a, c, d, e,$ dan F; komite D anggotanya adalah B, D, dan e; komite E anggotanya adalah B dan e. Apakah ada ***sistem perwakilan yang berbeda***, seorang wakil dari masing-masing komite, sehingga berbeda? yaitu sebuah himpunan yang memilih semua orang yang dipilih adalah untuk graf bipartit

6. Tunjukkan bagaimana masalah pencocokan kardinalitas maksimum dapat direduksi menjadi masalah aliran maksimum yang dibahas dalam Bagian 10.2.

7. Pertimbangkan algoritma serakah berikut untuk menemukan pencocokan maksimum dalam grafik bipartit: $G = V, U, E$. Urutkan semua simpul dalam urutan derajat yang tidak menurun. Pindai daftar yang diurutkan ini untuk menambahkan ke pencocokan saat ini (awalnya kosong) tepi dari simpul bebas daftar ke simpul bebas yang berdekatan dengan derajat terendah. Jika simpul daftar cocok atau jika tidak ada simpul bebas yang berdekatan untuknya, simpul tersebut dilewati begitu saja. Apakah algoritma ini selalu menghasilkan pencocokan maksimum pada graf bipartit?

8. Rancang algoritma waktu-linear untuk menemukan kecocokan maksimum di pohon.

9. Terapkan algoritme pencocokan maksimum bagian ini dalam bahasa pilihan Anda. Percobaan dengan kinerjanya pada grafik bipartit dengan n simpul di masing-masing set simpul dan tepi yang dihasilkan secara acak (di keduanya)

mode padat dan jarang) untuk membandingkan waktu berjalan yang diamati dengan efisiensi teoritis algoritme.

10. *Teka-teki domino* Domino adalah 2 × 1 ubin yang dapat diorientasikan baik secara horizontal maupun vertikal. Ubin papan yang diberikan terdiri dari 1× 1 kotak menutupinya dengan kartu domino dengan tepat dan tanpa tumpang tindih. Apakah mungkin untuk memasang kartu domino dengan angka 8× 8 papan tanpa dua unit persegi di sudut diagonal yang berlawanan?

10.4 Masalah Pernikahan Yang Stabil

Di bagian ini, kami mempertimbangkan versi menarik dari pencocokan bipartit yang disebut masalah pernikahan yang stabil. Pertimbangkan satu set *kamu* = {$M_1, M_2, \ldots, M_n$} dari *n* pria dan satu set *x* = {$w_1, w_2, \ldots, w_n$} dari *n* wanita. Setiap pria memiliki daftar preferensi yang memerintahkan wanita sebagai calon pasangan pernikahan tanpa ikatan diperbolehkan. Demikian pula, setiap wanita memiliki daftar preferensi pria, juga tanpa ikatan. Contoh dari dua set daftar ini diberikan pada Gambar 10.11a dan 10.11b. Informasi yang sama juga dapat disajikan oleh *n* × *n* matriks peringkat (lihat Gambar 10.11c). Baris dan kolom matriks masing-masing mewakili pria dan wanita dari dua himpunan. Sel dalam baris *M* dan kolom *w* berisi dua peringkat: yang pertama adalah posisi (peringkat) dari *w* dalam *M* daftar preferensi; yang kedua adalah posisi (peringkat) dari *M* dalam *w* daftar preferensi. Misalnya, pasangan 3, 1 pada baris Jim dan kolom Ann pada matriks pada Gambar 10.11c menunjukkan bahwa Ann adalah pilihan ketiga Jim sedangkan Jim adalah pilihan pertama Ann. Manakah dari dua cara untuk mewakili informasi tersebut yang lebih baik tergantung pada tugas yang ada. Misalnya, lebih mudah untuk menentukan kecocokan elemen himpunan dengan menggunakan matriks peringkat, sedangkan daftar preferensi mungkin merupakan struktur data yang lebih efisien untuk menerapkan algoritma pencocokan.

A ***pencocokan pernikahan*** *M* adalah sekumpulan *n (m, w)* pasangan yang anggotanya dipilih dari disjoint *n*-set elemen *kamu* dan *x* dengan cara satu-satu, yaitu, setiap orang *M* dari *kamu* dipasangkan dengan tepat satu wanita *w* dari *x* dan sebaliknya. (Jika kita mewakili *kamu* dan *x* sebagai simpul dari graf bipartit lengkap dengan tepi yang menghubungkan pasangan yang mungkin menikah, maka pencocokan pernikahan adalah pencocokan sempurna dalam grafik tersebut.)

preferensi pria

	1	ke-2	ke-3
Bob:	Lea	Ann	Sue
Jim:	Lea	Menuntut	Ann
Tom:	Menuntut	Lea	Ann

(A)

preferensi wanita

	1	ke-2	ke-3
Ann:	Jim	tom	Bob
Lea:	tom	Bob	Jim
Menuntut:	Jim	tom	Bob

(B)

matriks peringkat

	Ann	Lea	Menuntut
Bob	2,3	1,2	3,3
Jim	3,1	1,3	2,1
tom	3,2	2,1	1,2

(C)

GAMBAR 10.11 Data untuk contoh masalah pernikahan yang stabil. (a) Daftar preferensi pria; (b) daftar preferensi perempuan. (c) Matriks peringkat (dengan sel-sel kotak yang menyusun kecocokan yang tidak stabil).

Sepasang *(m, w),* di mana $M \in Y$, $w \in X$, dikatakan sebagai ***memblokir pasangan*** untuk jodoh *M* jika laki-laki *M* dan wanita *w* tidak cocok di *M* tapi mereka lebih suka satu sama lain daripada pasangan mereka di *M.* Misalnya, (Bob, Lea) adalah pasangan pemblokiran untuk pencocokan pernikahan *M* = {(Bob, Ann), (Jim, Lea), (Tom, Sue)} (Gambar 10.11c) karena mereka tidak cocok dalam *M* sementara Bob lebih menyukai Lea daripada Ann dan Lea lebih menyukai Bob daripada Jim. Sebuah pernikahan yang cocok *M* disebut ***stabil*** jika tidak ada pasangan pemblokiran untuk itu; sebaliknya, *M* disebut ***tidak stabil***. Menurut definisi ini, kecocokan pernikahan pada Gambar 10.11c tidak stabil karena Bob dan Lea dapat meninggalkan pasangan yang ditunjuk untuk bergabung dalam persatuan yang mereka berdua sukai. NS ***masalah pernikahan yang stabil*** adalah untuk menemukan kecocokan pernikahan yang stabil untuk preferensi yang diberikan pria dan wanita.

Anehnya, masalah ini selalu memiliki solusi. (Dapatkah Anda menemukannya untuk contoh pada Gambar 10.11?) Ini dapat ditemukan dengan algoritma berikut.

Algoritma pernikahan yang stabil

Masukan: Satu set *n* pria dan satu set *n* wanita bersama dengan peringkat wanita oleh setiap pria dan peringkat pria oleh setiap wanita tanpa ikatan yang diizinkan dalam peringkat

Output: Pencocokan pernikahan yang stabil

Langkah 0 Mulailah dengan semua pria dan wanita bebas.

Langkah 1 Meskipun ada orang bebas, pilih salah satu dari mereka secara sewenang-wenang dan lakukan mengikuti:

Usul Orang bebas yang dipilih *M* mengusulkan untuk *w,* selanjutnya wanita di daftar preferensinya (yang merupakan wanita dengan peringkat tertinggi yang belum pernah menolaknya sebelumnya).

Tanggapan Jika *w* gratis, dia menerima proposal untuk dicocokkan dengan *M*. Jika dia tidak bebas, dia membandingkan *M* dengan pasangannya saat ini. Jika dia lebih suka *M* kepadanya, dia menerima *M*lamaran, membebaskan mantan pasangannya; jika tidak, dia hanya menolak *M*usul, pergi *M*Gratis.

Langkah 2 Kembalikan himpunan *n* pasangan yang cocok.

Sebelum kita menganalisis algoritma ini, akan berguna untuk melacaknya pada beberapa input. Contoh seperti itu disajikan pada Gambar 10.12.

Mari kita bahas sifat-sifat algoritma perkawinan stabil.

DALIL Algoritme pernikahan yang stabil berakhir setelah tidak lebih dari n^2 iterasi dengan output perkawinan yang stabil.

BUKTI Algoritma dimulai dengan *n* laki-laki memiliki total n^2 wanita dalam daftar mereka. peringkat Pada setiap iterasi, seorang pria mengajukan lamaran kepada seorang wanita. Ini mengurangi jumlah total wanita yang masih dapat dilamar oleh pria di masa depan karena tidak ada pria yang melamar wanita yang sama lebih dari satu kali. Oleh karena itu, algoritma harus berhenti setelah tidak lebih dari n^2 iterasi.

GAMBAR 10.12 Penerapan algoritma pernikahan yang stabil. Proposal yang diterima adalah ditunjukkan oleh sel kotak; proposal yang ditolak ditunjukkan oleh sel yang digarisbawahi.

Mari kita buktikan bahwa pencocokan akhir M adalah pasangan pernikahan yang stabil. Karena algoritma berhenti setelah semua n laki-laki cocok satu-satu dengan n wanita, satu-satunya hal yang perlu dibuktikan adalah stabilitas *M.* Misalkan, sebaliknya, bahwa M tidak stabil. Lalu ada sepasang pria yang menghalangi M dan seorang wanita w yang tak tertandingi dalam M dan sedemikian sehingga keduanya M dan w lebih memilih satu sama lain daripada orang yang cocok dengan mereka *M.* Sejak M melamar setiap wanita di daftar peringkatnya dalam urutan preferensi yang menurun dan w mendahului M pertandingan di *M, saya* pasti sudah melamar w pada beberapa iterasi. Apakah w menolak M atau menerimanya tetapi menggantikannya pada iterasi berikutnya dengan pertandingan berperingkat lebih tinggi, w jodoh di M harus lebih tinggi w daftar preferensi daripada M karena peringkat pria yang dicocokkan dengan wanita tertentu hanya dapat meningkat pada setiap iterasi algoritme. Ini bertentangan dengan asumsi bahwa w lebih suka M ke pertandingan terakhirnya di M. ■

Algoritme pernikahan yang stabil memiliki kekurangan yang mencolok. Itu bukan "netral gender." Dalam bentuk yang disajikan di atas, itu lebih menyukai preferensi pria daripada wanita

preferensi. Kita dapat dengan mudah melihat ini dengan menelusuri algoritma pada contoh masalah berikut:

	wanita 1	wanita 2
pria 1	1, 2	2, 1
pria 2	2, 1	1, 2

Algoritme jelas menghasilkan pencocokan yang stabil M = {(pria 1, wanita 1), (pria 2, wanita 2)}. Dalam pencocokan ini, kedua pria dicocokkan dengan pilihan pertama mereka, yang tidak terjadi pada wanita. Seseorang dapat membuktikan bahwa algoritma selalu menghasilkan pencocokan yang stabil yaitu***man-optimal***: itu memberikan kepada setiap pria wanita peringkat tertinggi yang mungkin di bawah pernikahan yang stabil. Tentu saja, bias gender ini dapat dibalik, tetapi tidak dihilangkan, dengan membalikkan peran yang dimainkan oleh laki-laki dan perempuan dalam algoritme, yaitu dengan membuat perempuan melamar dan laki-laki menerima atau menolak lamaran mereka.

Ada konsekuensi penting lainnya dari fakta bahwa algoritma pernikahan yang stabil selalu menghasilkan pencocokan stabil yang optimal gender. Sangat mudah untuk membuktikan bahwa pencocokan optimal pria (wanita) adalah unik untuk serangkaian preferensi peserta tertentu. Oleh karena itu keluaran algoritma tidak tergantung pada urutan laki-laki (perempuan) bebas membuat proposal mereka. Akibatnya, kita dapat menggunakan struktur data apa pun yang kita inginkan—misalnya, antrian atau tumpukan—untuk merepresentasikan set ini tanpa berdampak pada hasil algoritme.

Gagasan tentang pencocokan stabil serta algoritma yang dibahas di atas diperkenalkan oleh D. Gale dan LS Shapley dalam makalah berjudul “Penerimaan Perguruan Tinggi dan Stabilitas Pernikahan” [Gal62]. Saya tidak tahu mana dari dua aplikasi yang disebutkan dalam judul yang Anda anggap lebih penting. Intinya adalah bahwa stabilitas adalah properti yang cocok yang dapat diinginkan dalam berbagai aplikasi. Misalnya, telah digunakan selama bertahun-tahun di Amerika Serikat untuk mencocokkan lulusan sekolah kedokteran dengan rumah sakit untuk pelatihan residensi. Untuk sejarah singkat aplikasi ini dan diskusi mendalam tentang masalah perkawinan yang stabil dan perluasannya, lihat monografi Gusfield dan Irwing [Gus89].

Latihan 10.4

1. Pertimbangkan contoh matriks peringkat masalah pernikahan yang stabil: diberikan oleh berikut

	A	B	C
α	1, 3	2, 2	3, 1
β	3, 1	1, 3	2, 2
γ	2, 2	3, 1	1, 3

Untuk setiap jodohnya, tunjukkan apakah itu stabil atau tidak. Untuk pencocokan yang tidak stabil, tentukan pasangan pemblokiran. Untuk pencocokan yang stabil, tunjukkan apakah mereka optimal untuk pria, optimal untuk wanita, atau tidak keduanya. (Asumsikan bahwa huruf Yunani dan Romawi masing-masing menunjukkan pria dan wanita.)

2. Rancang algoritma sederhana untuk memeriksa apakah pencocokan pernikahan yang diberikan stabil dan tentukan kelas efisiensi waktunya.

3. Temukan kecocokan pernikahan yang stabil untuk contoh yang diberikan dalam Soal 1 dengan menerapkan algoritma pernikahan yang stabil

A. dalam versi melamar pria.

B. dalam versi yang melamar wanita.

4. Temukan matriks peringkat pencocokan untuk contoh yang didefinisikan sebagai berikut: pernikahan yang stabil:

	A	B	C	D
α	1, 3	2, 3	3, 2	4, 3
β	1, 4	4, 1	3, 4	2, 2
γ	2, 2	1, 4	3, 3	4, 1
δ	4, 1	2, 2	3, 1	1, 4

5. Tentukan kelas efisiensi waktu dari algoritma perkawinan stabil

A. dalam kasus terburuk.

B. dalam kasus terbaik.

6. Buktikan bahwa set pernikahan stabil yang optimal untuk pria selalu unik. Apakah ini juga berlaku untuk pencocokan pernikahan stabil yang optimal bagi wanita?

7. Buktikan bahwa dalam pencocokan stabil pria-optimal, setiap wanita memiliki pasangan terburuk yang bisa dia miliki dalam pencocokan pernikahan yang stabil.

8. Implementasikan algoritma pernikahan stabil yang diberikan di Bagian 10.4 sehingga waktu berjalan masuk $Pada_2)$. Jalankan eksperimen untuk memastikan efisiensi kasus rata-ratanya.

9. Tulis laporan tentang ***masalah masuk perguruan tinggi*** (tugas penghuni-rumah sakit) yang menggeneralisasi masalah pernikahan yang stabil di mana sebuah perguruan tinggi dapat menerima "proposal" dari lebih dari satu pelamar.

10. Pertimbangkan ***masalah teman sekamar***, yang terkait dengan tetapi lebih sulit daripada masalah pernikahan yang stabil: “Sejumlah anak laki-laki yang genap ingin dibagi menjadi pasangan teman sekamar. Seperangkat pasangan disebut stabil jika di bawahnya tidak ada dua anak laki-laki yang bukan teman sekamar dan yang lebih memilih satu sama lain daripada teman sekamar mereka yang sebenarnya.” [Gal62] Berikan contoh masalah ini yang *bukan* memiliki pasangan yang stabil.

RINGKASAN

- NS *teknik perbaikan berulang* melibatkan menemukan solusi untuk masalah optimasi dengan menghasilkan urutan solusi yang layak dengan meningkatkan nilai-nilai fungsi tujuan masalah. Setiap solusi berikutnya dalam urutan seperti itu biasanya melibatkan perubahan kecil yang terlokalisasi dalam solusi layak sebelumnya. Ketika tidak ada perubahan seperti itu yang meningkatkan nilai

fungsi tujuan, algoritma mengembalikan solusi layak terakhir sebagai optimal dan berhenti.

- Masalah penting yang dapat diselesaikan secara tepat dengan algoritma perbaikan iteratif termasuk pemrograman linier, memaksimalkan aliran dalam jaringan, dan mencocokkan jumlah maksimum yang mungkin dari simpul dalam grafik.
- NS *metode simpleks* adalah metode klasik untuk memecahkan masalah program linier umum. Ini bekerja dengan menghasilkan urutan titik ekstrim yang berdekatan dari wilayah layak masalah dengan meningkatkan nilai fungsi tujuan.
- NS *masalah aliran maksimum* meminta untuk menemukan aliran maksimum yang mungkin dalam jaringan, grafik berarah berbobot dengan sumber dan wastafel.
- NS *Metode Ford-Fulkerson* adalah template klasik untuk memecahkan masalah aliran maksimum dengan pendekatan iterative-improvement. NS*metode shortestaugmenting-path* mengimplementasikan ide ini dengan melabeli simpul jaringan dengan cara pencarian luas-pertama.
- Metode Ford-Fulkerson juga menemukan *potongan minimum* dalam jaringan tertentu.
- A *pencocokan kardinalitas maksimum* adalah himpunan bagian terbesar dari sisi dalam graf sedemikian rupa sehingga tidak ada dua sisi yang berbagi titik yang sama. Untuk graf bipartit, dapat ditemukan dengan urutan augmentasi kecocokan yang diperoleh sebelumnya.
- NS *masalah pernikahan yang stabil* adalah untuk menemukan *pencocokan stabil* untuk elemen dua *n*- set elemen berdasarkan preferensi pencocokan yang diberikan. Masalah ini selalu memiliki solusi yang dapat ditemukan oleh*Algoritma Gale-Shapley*.