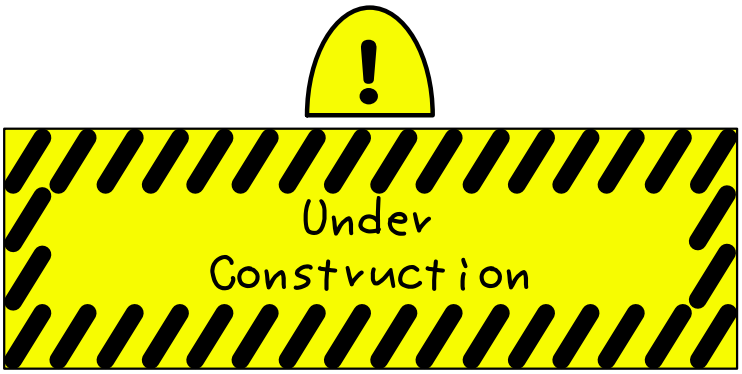
*dot*を使ったグラフ描画

Emden Gansner and Eleftherios Koutsofios and Stephen North
とみながだいすけ訳

February 4, 2003
April 28, 2004 訳

概 要

dot はグラフを階層的に描くツールである。コマンドラインで、あるいは web アプケーションとして、また GUI を通じて使うことができる。その特徴として、ノードやエッジ曲線、エッジのラベルを配置する優れたレイアウト・アルゴリズム、データ構造を描くための「ポート」を持つ「レコード」型、レイアウトのクラスタリング、ストリーム入出力を利用するグラフツールとしてのファイル言語などがある。以下は SML-NJ コンパイラの簡約化した依存関係だが、1.4GHz の AMD Athron プロセッサで 0.98 秒で描くことができる。



1 はじめに

dot は有効グラフを描画するプログラムである。様々な属性情報を持った、グラフを記述したテキストファイルを読み込み、グラフを記述したファイル、または GIF、PNG、SVG、PostScript などの画像ファイルとしてグラフを出力する。

dot によるグラフの描画は 4 段階からなる。これを知ることで、*dot* が行うレイアウトの種類、またレイアウトの指定の仕方などを理解する助けになるだろう。*dot* が行うレイアウト法は、グラフが環状ではないことを前提にしている。したがって最初のステップは、入力されるグラフの記述中で、環状の部分に含まれるエッジの向きを変更することで、全ての環を開くことである。次のステップは各ノードに離散的なランクあるいはレベルを割り当てる。上から下向きの描画において、ランクは Y 座標を決めることになる。一つ以上のランクをとばすエッジは、単位長さを持つ「仮想的」なエッジの鎖として分割される。三番目のステップは、各ランク内でエッジが交叉しないようにノードを並べることである。四番目のステップはエッジが短くなるように各ノードの X 座標を決めることで、最後のステップでエッジをスプライン補間で描画する。こういった方法は階層的なグラフを描くプログラムで一般的に用いられており、Warfield[War77]、Carpano[Car80]、Sugiyama[STT81] などを元にしている。

dot は *DOT* 言語（付録 A 参照）による入力を受け付ける。この言語ではグラフ、ノード、エッジの三種類のオブジェクトについて記述できる。メインとなる（全体の）グラフは有向グラフ（`digraph`）または無向グラフ（`graph`）である。*dot* はグラフを有向グラフとしてレイアウトするため、今後例示するグラフは有向グラフとする（レイアウトを行うユーティリティ *neato* で無向グラフを描くことができる [Nor92]）。メインのグラフの中で、ノードやエッジのサブセットを `subgraph` として定義できる。

図 1 に *DOT* 言語によるグラフの例を示す。1 行目はグラフの名前をタイプを定義している。それに続く行ではノード、エッジ、サブグラフを生成し、属性を指定している。これらのオブジェクトの名前には C 言語の識別子、数字、クオーテーションされた C 言語の文字列を使うことができる。クオーテーション中では句読点やスペースを使うことができる。

ノードは、ファイル中でその名前が最初に現れたときに生成される。エッジは、エッジ演算子`->`でノードが結ばれたときに生成される。この例では 2 行目でノード *main* から *parse*、*parse* から *execute* へのエッジが作られている。以下のようにして、このファイル（`graph1.dot` とする）を *dot* に入力すると、

```
$ dot -Tps graph1.dot -o graph1.ps
```

図 2 に示す画像が得られる。`-Tps` というコマンドライン・オプションは出力形式に PostScript（EPSF）形式を指定する。`graph1.ps` は PostScript を表示できるアプリケーションなどで表示、印刷したり、他の文書に取り込むことができる。

ノードやエッジの外見や位置を調整したいと思うこともあるが、入力ファイル中でノード、エッジ、サブグラフの属性を指定することでそれができる。属性は属性名とその値の対として指定する。図 3、4 にその例を示す。図 3 のリスト中で、2 行目ではグラフのサイズ（`size`）を `4, 4`（インチ）に指定している。この属性で描画するグラフの大きさを指定できる。もし描画されたグラフがこの指定よりも大きくなった場合には、指定に合うように縮小される。

ノードやエッジの属性は、角括弧の中で指定される。3 行目で、`main` というノードの形を `box` に指定している。4 行目では `weight` を大きく（デフォルトの値は 1）することでエッジが曲がらないようにしている。6 行目で描かれるエッジは破線である。8 行目では `execute` から `make_string`

と printf へのエッジを生成している。10 行目ではエッジの色のデフォルトを赤 red にしている。これはこのファイル中でこの行よりも後の部分に対して有効となる。11 行目ではエッジを太くし、100 times というラベルを付けている。12 行目では make_string というノードに複数行のラベルを付けている。13 行目ではノードをデフォルトを長方形で青い色で塗るように指定している。ノード compare ではそういった指定が適用される。

2 描画属性

グラフ描画のための全ての属性のリストを表 1、2、3 に示す。

2.1 ノードの形

デフォルトではノードは shape=ellipse,width=.75,height=.5 として描かれ、ラベルにその名前を用いる。形には他に box、circle、record、plaintext がある。付録 E にすべてを列挙している。plaintext ではノードの外枠を描かないため、ある種の図ではこれが重要であることもある。グラフの構造のみが重要であり、グラフがある程度大きくなる場合、形に point を指定するとノードを最小にできる。描画を行うとき、ノードはラベル文字列を配置するのに必要な大きさにまで大きくされるが、fixedsize=true を指定して width と height を固定するとそうはならない。

ノードの形は、多角形タイプとレコードタイプ[1]の二種類に大別される。record と Mrecord 以外の形は全て多角形として捉えられ、辺の数（円と楕円はその特例と考える）と座標値などで表される。そのうちいくつかはグラフ中で指定できる。regular=true を指定すると、ノードの形は正多角形になる。peripheries で辺を何重にするかを指定できる。たとえば peripheries=2 で二重丸になる。orientation 属性でノードの傾きを時計回りの向きの度で指定できる。

形を polygon にすると多角形を定義する全てのパラメータを使って、既定の形以外の様々な形を作ることができる。前述した regular、peripheries、orientation の他に辺の数 sides、skew、distortion を使って多角形の形を指定することができる。skew は浮動小数値（普通は-1.0 から 1.0 の間）で、上から下方向に多角形を傾けて歪めることを指定する。値が正なら多角形の上部を右に動かすように歪む。したがって skew を使うと長方形から平行四辺形を作ることができる。distorion は多角形を上から下方向に縮小する。値が負なら下部が上部よりも大きくなる。distortion は長方形を台形にする。こういった多角形のための属性の様々な例を図 5、6 に示す。

レコードタイプのノードはまた異なった形である。このタイプには record と Mrecord がある。後者は角が丸められていることを除くと、この二者は同じである。このタイプのノードは再帰的なフィールドのリストで、長方形領域の並びを水平方向、垂直方向と交互に繰り返す。再帰構造はノードのラベル label で指定され、以下の文法を持つ。

$$
\begin{aligned}
rlabel &\rightarrow field('|'field)* \\
field &\rightarrow boxLabel|''rlabel'' \\
boxLabel &\rightarrow ['<' string '>'][string]
\end{aligned}
$$

角括弧、縦棒、三角括弧はエスケープする必要がある。空白はトークン間のセパレータとして認識され、テキスト中に空白を使いたいときにはやはりエスケープする必要がある。*boxLabel* 中の最

[1] shape=epsf と shapefile で自由にノードの形を実装し、PostScript 出力に用いることができる。詳細についてはこのガイドの範疇ではないが、著者に連絡をいただきたい。

初の文字列はそのフィールドの名前となり、その長方形領域のポートの名前となる（3.1 章参照）。二番目の文字列はそのフィールドのラベルになり、ラベルを複数行にしたいときにはエスケープ・シーケンスを用いる（2.2 章参照）。レコードの属性のうちのいくつかを使った例を図 7、8 に示す。

2.2 ラベル

ラベルの名前はデフォルトではその名前が用いられることを上述した。エッジにはデフォルトではラベルは付かない。図 4 に示すように、ノードにもエッジにも `label` を使って明示的に指定することでラベルを付けることができる。

ノードの名前をラベルに用いるのは便利であると考えられるが、そうでない場合には指定しなければならない。たとえばファイルのディレクトリ・ツリーを描画したい場合、`src` という名前のディレクトリが複数あっても、それぞれ別のノード名を持つ必要がある。inode 番号やフルパスが適当であろう。ディレクトリ中のファイル名なら、そのままノード名に使うことができる。

複数行からなるラベルはエスケープ・シーケンスの`\n`、`\l`、`\r` を行末に置くことで作ることができ、それぞれ各行をセンタリング、左寄せ、右寄せすることになる[2]。

ノードの形 `Mdiamomnd`、`Msquare`、`Mcircle` では、`toplabel` および `bottomlabel` という属性を、それぞれノードの上部、下部に付けるラベルを指定するのに使うことができる。

グラフおよびサブグラフ（クラスタ）にもラベルを付けることができる。グラフのラベルは、デフォルトではグラフの下部中央に付く。`labelloc=t` を設定するとラベルはグラフの上部中央に付く。クラスタのラベルは長方形で囲まれた中、上部左端に配置される。`labelloc=b` を設定すると、ラベルは長方形内の下部に付く。`labeljust=r` とすると右寄せになる。

デフォルトで使われるフォントは 14 ポイントの Times-Roman で、色は黒である。`fontname`、`fontsize`、`fontcolor` 属性を指定することで、他のフォントファミリ、フォントサイズ、文字の色を使うことができる。どのようなグラフィクス言語でも用意されているフォントファミリ、Times、Helvetica、Courier、Symbol を使うようにするべきである。たとえば `Times-Italic`、`Times-Bold`、`Courier` などはトラブルが少ないが、`AvanteGarde-DemiOblique` などといったものは避ける方がよい。

dot が GIF や JPG などのビットマップ出力をする際、レイアウト中にこれらのフォントが使えるようになっていなければならない。フォントファイルを検索するためのディレクトリのリストを、`fontpath` 属性で指定することができる[3]。これが設定されないときは *dot* は環境変数の DOTFONTPATH を使うが、それも設定されていないときは GDFONTPATH を使う。どちらも設定されていないときは、*dot* に組み込まれているリストを用いる。

エッジのラベルは、エッジの中心あたりに付けられる。ラベルはエッジやノードに重ならないように配置される。しかし複雑なグラフになると、どのラベルがどのエッジのものか分からなくなるようなこともある。`decorate` 属性を true に設定すると、ラベルとエッジ結ぶ線が引かれる。エッジトラベルが重なるのを防ぐため、グラフが思ったりも大きくなってしまうこともある。`labelfloat=true` を設定すると、そういった重なりを許してよりコンパクトなグラフを生成する。

`headlabel` および `taillabel` を使って、エッジに付けるラベルを増やすことができる。こういったラベルは、`labelfontname`、`labelfontsize`、`labelfontcolor` 属性で外見を変更できる。これらのラベルはエッジとノードの交点の近くに配置され、エッジやノードと重なってしまうこともある。`labelangle`、`labeldistance` 属性を使うとそれを調整することができる。前者はノードにつ

[2] エスケープ・シーケンスの`\N` は、ノード名を表す内部シンボルである。
[3] Unix 系のシステムではパス名をコロンで区切って連結したもの。Windows 系ではセミコロンで区切ったものである。

ながっているエッジとラベルのなす角度を度で指定する。後者はラベルとノードの距離をスケーリングする倍数の数値である。

2.3 スタイル

ノードとエッジには color 属性を指定できる。デフォルトは黒である。これはノードの輪郭やエッジを描くときの色になる。color は色、彩度、明度の三つをそれぞれ 0 から 1 までの実数で、コンマで区切って並べたものか、付録 G に挙げる色名（X Window System のどれかのバージョンと同じ）か、赤、緑、青（RGB）の強さをそれぞれ'#' に続く 00 から FF までの 16 進数で表して並べたもののいずれかである。したがって色の指定は"orchid"、"0.8396,0.4862,0.8549"、#DA70D6 のような形になり、この場合は三つとも同じ色を指定している。数値で表現する方法はスクリプトや何かのツールで色を自動で生成するときに便利だろう。色名では英字の大文字、小文字は区別せず、英数字以外は無視される。つまり warmgrey と Warm_Grey は同じになる。

グラフを描画する上での色の使い方には、いくつか注意した方がよいことがある。まず、明るい色をあまり多く使うのは避けるべきである。いわゆる「レインボー・エフェクト」は混乱しがちである。色の範囲をあまり広くしない、または同じ色で彩度を変えて用いるのがよい。次に、ノードを暗い色または彩度の高い色で塗りつぶす場合、ラベルは fontcolor=white および fontname=Helvetica とすると読みやすい（プレイン・フォントからアウトライン・フォントを生成する PostScript の機能を *dot* から使うことができる）。三つ目は、出力フォーマットによってはユーザー独自の色空間を定義できることである。たとえば PostScript 出力の場合、nodecolor、edgecolor、graphcolor をライブラリ・ファイルの中で定義しなおすことができる。RGB で色を使いたい場合は以下の行を lib.ps というファイルに記入する。

```
/nodecolor setrgbcolor bind def
```

そしてこのファイルを読み込むために-l オプションをコマンドラインで指定する。

```
dot -Tps -l lib.ps file.dot -l file.ps
```

style 属性はノードとエッジの様々な外見を変更するために使われる。これはコンマで区切って要素を並べたリストで、オプションで引数を与えることもできる。あらかじめ定義されている要素には solid、dashed、dotted、bold、invis がある。最初の四つはノードの輪郭やエッジを描くときに使われ、それぞれ言葉通りの意味である。invis はノードやエッジを描かないように指定する。ノードのスタイルは他に filled、diagonals、rounded が指定できる。filled は fillcolor で指定された色でノードの輪郭内を塗りつぶす。fillcolor が指定されていないときは color が使われる。これも指定されていないときは明るい灰色 light grey がデフォルト値として使われる[4]。diagonals は辺の対について、頂点付近から短い対角線を引く。rounded は多角形の角を丸める。

PostScript 手続きを使えば、ユーザーが独自のスタイル要素を定義できる。そういった要素はグラフ、ノード、エッジの gsave コンテキスト内で、まだなにも描画されていないときに実行される。引数のリストは PostScript 文法に合わせて変換される。たとえば、style="setlinewidth(8)"をあるノードに指定すると、輪郭が太い線で描かれる。ここで setlinewidth は PostScript の組み込みだが、ユーザーが定義した手続きも同様に呼び出すことができる。ユーザーによる定義はライブラリ・ファイル中に記述され、上述した-l を使った方法で読み込む。

[4]出力フォーマットが MIF のとき、またはノードの形が point のときはデフォルトは黒 black になる。

エッジには dir 属性で矢印を付けることができる。dir は forward（デフォルト値）、back、both、none のいずれかである。これは矢印の頭を描くときにだけ参照され、グラフのレイアウトには影響を与えない。たとえば dir=back を指定するとエッジの終端に矢印が描かれ、先端には描かれない。しかしエッジの端点を入れ替えるわけではない。arrowhead および arrowtail 属性はそれぞれエッジの先端と終端に描く矢印のスタイルを指定する。normal、inv、dot、invdot、odot、invodot、none のいずれかが指定できる（付録 F 参照）。arrowsize は、そのエッジに描かれる矢印の大きさを拡大、縮小する倍数を指定する。たとえば arrowsize=2.0 は矢印の長さを 2 倍に、幅も 2 倍にする。

スタイルと色については、クラスタは大きな長方形のノードのように扱われる。クラスタの外枠はそのクラスタの color 属性で色が付けられ、一般的にクラスタの外見は sytle、color、fillcolor 属性で変更できる。

一番外側のグラフ（root グラフ）には bgcolor という属性があり、グラフ全体の背景色を指定し、これはまた塗りつぶしの色のデフォルト値としても使われる。

2.4 描画の向き、サイズ、余白

dot が描画を行う際に、グラフの大きさを決めるのに重要な属性が、nodesep と ranksep である。前者は同じランク内で隣り合うノード間の距離の最小値をインチで指定する。後者はランク間の距離で、あるランクのノードの下端から次のランクのノードの上端までの上下方向の距離の最小値である。ranksep 属性はランク間の距離をインチで指定する。または、ranksep=equally という指定もできる。これにより全てのランク間の距離が同じになるように配置される。このときランク間の距離は、そのランク中のノードの中心から、他のランクのノードの中心までである。この場合二つのランク間の距離は最も小さい場合でデフォルト値になる。 ranksep の指定法は二種類あるが、これらを同時に指定することもできる。たとえば ransep="1.0 equally"とするとランク間の距離はすべて同じになり、その距離は 1 インチ以下にはならない。

出力先のプリンタや、図を取り込もうとする文書によっては、ノードのサイズや間隔がデフォルト値のままだと図が大きくなりすぎることもあるだろう。この問題への対処法はいくつかある。まず最初に、*dot* が計算する最終的な図の大きさはどのように決められるのかを見てみる。

レイアウトの大きさは、内部ではデフォルト値を使ってまず「自然な」大きさに設定される（後述する ratio=compress が設定されていない場合）。図の大きさや縦横比は設定されないため、もしグラフが大きくなった場合は、図も大きくなる。size も ratio も指定されないときは、その自然な大きさで出力される。

図の大きさを指定する最も簡単な方法は size="x,y"をグラフのファイルで指定する（またはコマンドライン・オプション-G で指定する）ことである。これにより最終的な図の大きさが指定される。たとえば size="7.5,10"とすると 8.5x11 サイズの用紙に合う（用紙の向きをデフォルトのままと仮定した場合）。レイアウトの初期段階でどんなに図が大きくなっても、問題なく出力される。

ratio も図の大きさに影響を与える。size と ration の設定に応じて、いくつかの場合が考えられる。

Case 1. ratio が設定されていない場合。レイアウトが与えられた size にあっている場合は、なにも行われない。そうでなければ指定された大きさに収まるまで、縦横比を保って縮小される。

ratio が設定されている場合は、四つの場合がある。

Case 2a. ratio=x（x は実数値）が指定されている場合は高さ/幅で表される比が与えられた比 ratio になるまで、一方向に沿って図が拡大される。たとえば ratio=2.0 とすると図の高さが幅の

2 倍になる。それから Case 1 と同様に、与えられた size になるまで拡大・縮小される。

Case 2b. ratio=fill および size=x,y が指定されているとき、比が y/x になるまでレイアウトは一方向に沿って拡大され、それから Case 1 同様に拡大・縮小される。size で指定する領域の大きさになるまで拡大・縮小される。**Case 2c.** ratio=compress および size=x,y が指定されているとき、初期段階のレイアウトは与えられた大きさになるよう縮小される。これにより、レイアウトの品質、バランス、対称性などを犠牲にして小さくまとめられたレイアウトになる。そして Case 1 と同様に拡大・縮小される。

Case 2d. ratio=auto および page 属性が設定され、グラフが 1 ページに収まらない場合、size 属性は無視され *dot* は「理想的」な大きさを計算する。特に、初期レイアウトの大きさの半分よりもページの大きさが大きく、ページの大きさを整数倍すると与えられたサイズを超えてしまうような場合である。縦の横の二方向でそれぞれ独立に新しい大きさが計算される。

rotate=90 または orientation=landscape が指定されている場合、図は 90 度回転してランドスケープ・モードになる。図の X 軸は各ページで Y 軸の値になる。これによって dot の size、ratio、page の解釈が影響を受けることはない。

page が指定されていない場合、出力は 1 ページになる。

page=x,y と指定されている場合、図はタイル、あるいはモザイク状に分割され、複数ページに出力される。よく用いられる設定は page="8.5,11"や page="11,17"である。これらの値は出力するプリンタの最大出力サイズを参照し、実際の出力サイズは余白の設定により縮小する（プリンタ用の設定では 0.5 インチ、ビットマップ出力では X、Y それぞれに対して 10 および 2 ポイントである）。タイル状のレイアウトの場合は余白は小さい方がよい。これは margin 属性で設定できる。一つの数値で両方の余白を設定するか、コンマで区切った二つの数値で x と y の余白をそれぞれ指定するか、どちらかである。普通は単位はインチである。margin=0 という設定もできるが、多くのビットマッププリンタは内部でハードウェア的に余白が設定されており、これを無効化することはできない。

ページが出力される順序を pagedir 属性で設定できる。出力は常に行ごと、あるいは列ごとに行われるが、pagedir を二文字で設定することで第一の順序と第二の順序をそれぞれ指定できる。たとえばデフォルト値は BL だが、これは第一の順序を下から上方向（bottom-to-top、B）、第二の順序を左から右方向（left-to-right、L）に指定する。したがってまず最下行のページが左から右に出力されていき、次に下から 2 番目の行が左から右に、その淳で一番上の行まで出力される。上から下方向は T、右から左方向は R で指定できる。

center=true が指定され、グラフが 1 ページに収まるとき、page が指定されていなければデフォルトのサイズ 8.5x11 インチが適用され、図はページの中央に配置される。

大きなグラフを小さなサイズに入れようとすると、ノードのラベルの字が小さくて見えないという問題が生じる。ラベルを大きくするにはそういう設定が必要である。1 ページ中で読むことのできる文字の量には限界がある。*dot* を実行する前に、元のグラフから重要な部分だけを抜き出すこともできるだろう。そのための便利なツールもある。

sccmap 結びつきが強い要素だけを抜き出す。

tred transitive reduction を計算する（transitivity に示されるエッジを削除する）。

unflatten ツリー構造の葉のエッジを揺さぶることで、縦横比を改善する。

また、以下のようなことを試してみると良い。

1. ノードの fontsize を大きくする。
2. ranksep や nodesep を小さくする。
3. ratio=auto にしてみる。
4. 適切な size を与えて ratio=compress を設定する。
5. 図が小さい場合は Times よりもサンセリフ体（Helvetica など）の方が見やすい。

2.5 ノードとエッジの配置

dot では様々な属性を使うことで、ノードトエッジの大規模なレイアウトを様々な方法で行うことができ、ユーザーの要求に応える品質の画像を得ることができる。この章ではそういった属性について述べる[5]。

時には、エッジの方向が上下ではなく左から右であった方が自然な図もある。rankdir=LR をグラフのトップ・レベルで指定すると、グラフがそういった向きで描かれる。デフォルトでは TB（top-to-bottom）である（BT という向きも下から上方向に描くのに便利なように思えるが、実装されていない。同様なことはエッジの端点を入れ替えること、つまり dir=back を設定することでできる）。orientation や rotate で図の向きが変えられる場合、rankdir が補足的な働きをすることになる。

時間軸を持つグラフや、ソースとシンクのノードを強調したい場合、ランクの割り当てを制限すると良いことがある。サブグラフの rank は samerank、minrank、source、maxrank、sink で指定できる。samerank の値は、そのサブグラフ内のノードを全て同じランク内に入れる。これが minrank に設定されていれば、そのサブグラフ内のノードは全て、その図の中で最小のランクのノードと同じランクになる[6]。rank=source を指定すると、そのサブグラフ内のノードのランクは他のどのノードよりも小さな値が設定される（他に minrank や source が指定されて多サブグラフを除く）。maxrank や sink という値は、最大ランクを指定するのと同様である。こういった制限を加えることで、ノードのクラスが同地になることがある。あるサブグラフでノード A と B が同じランク、他のサブグラフでノード C と D が同じランクだと設定すると、両サブグラフ内のノードは全て同じランクとして描かれる。図 9 と図 10 にランクを指定する例を挙げる。

グラフによってはノードを左から右に並べることが重要な場合がある。ordering=out をサブグラフで指定すると、同じサブグラフ内にあるエッジのうち、終端が同じノードにつながるものは、エッジの生成された順に左から右へと扇状に並べられる（またあるノードに始点がある複数のエッジは並行に並べられ、互いに重なる可能性がある）。

ノードトエッジのレイアウトは、うまく調整する方法がいくつもある。たとえばあるエッジの両端にあるノードの group 属性値が同じであれば、*dot* はできるだけ、そのエッジを直線にし、他のエッジと交差しないようにする。weight を指定してもエッジを直線にできる。wegith はエッジの重要性を表すような値で、大きな値（重み）を持つエッジはできるだけ短く、曲がらないように描かれる。

エッジの重みは、いくつかのノードが同じランクに指定されているときにも影響を持つ。そういったノード間に重みが 0 でないエッジがある場合、複数のランク間で同じ方向（左から右方向、

[5] より正確には、*dot* ではレイアウト・アルゴリズム中で技術的な役割を担う様々なパラメータを操作することができる、ということである。そういったパラメータには mclimit、mslimit、nslimit1、remincross、searchsize がある。

[6] 言い換えると、最小ランクということは図の一番上に描かれると言うことである。

ランドスケープでは上から下方向）になるものとされる。不可視のエッジ（style="invis"）を配置してノードの並び順を操作する必要があるようなときに、これを利用できる。

あるノードに複数のエッジの端点があるとき、samehead および sametail 属性を使うことができる。同じ矢印で samehead の値が同じエッジの端点はすべて、ノードの輪郭線上の一点につなげられる。エッジの終端でも sametail で同様のことができる。

ランクをノードに与えていくとき、エッジの先端のノードは終端のノードよりも高いランクを持つように設定される。しかしそのエッジに constraint=false が指定されている場合は、かならずしもそうはならない。

環境によっては、エッジの両端の距離があまり短くならないようにしたい場合もある。これは minlen 属性で指定できる。これはエッジの先端と終端のノードについて、そのランク間の差の最小値を指定する。minlen=2 を指定すると、先端と終端の間に少なくとも一つのランクが挿入される。これは二つのノード間の物理的な距離を指定できるわけではない。

こういった調整は注意深く行う必要がある。*dot* は、各ノードやエッジの配置についての操作やユーザーからの介入があまり多くないときに、一番うまくレイアウトできるようになっている。エッジの weight を増やしたり、不可視のノードやエッジを syte=invis で作ったり、ファイル中でノードやエッジの順序を変えたりすることでレイアウトを調整することができる。しかし、入力されるグラフ中での変更に関してレイアウトが不安定な部分もあり、かえって逆効果になることもある。どこか一カ所を調整してみただけで、それまでの全ての調整がムダになり、図のできばえが台無しになることもある。現在計画中ではあるが、ユーザーが指定するレイアウトの方法などが使える対話的操作が可能な入出力システムと、*dot* の数学的なレイアウト法を組み合わせることを将来行いたい。

3 優れた特徴

3.1 ノードのポート

ノードのポートとは、エッジがノードにつなげられる点のことである（エッジがポートにつながらない場合は、ノードの中心につなげられ、ノードの輪郭内の部分はクリッピングされる）。

headport および tailport 属性で単純ポートを指定できる。属性値には 8 方位、"n"、"ne"、"e"、"se"、"s"、"sw"、"w"、"nw"のいずれかが指定できる。エッジの端点はノード上の指定された点につながれる。つまり tailport=se とするとエッジの終点がノードの南西方向の「頂点」（右下）につながれる。

record タイプのノードでは、レコードの構造をポートの指定に用いる。前述したとおり、このタイプは長方形の再帰的なリストである。各長方形においてラベル中の<*port_name*>の形でポートを定義すると、その長方形の中心をポートとして使うことができる（デフォルトではエッジは長方形の輪郭線でクリッピングされる）。これは、エッジの宣言中で *node_name*:*port_name* の形でノード名にポート名を付け加えることで指定できる。図 11 にレコードタイプのノードでのポート姪を使った宣言を、図 12 に描画例を示す。

要注意： 単純ポートは機能はしても、現在ではまだ正式なものではない。たとえば tailport=n で headport=s のような指定はできない方がよいと考えている。そして理論的には、これらの考えは直行しており、一つのエッジについて両方のタイプのポートを使えるようにするべきだろう。二種類の指定法をどのように組みあわせるか、たとえば方角を表す文字列をポート名の予約語にし、nodename:portname:compassname のような表記ができるようにするか、という問題も

ある。

図 13 と図 14 に、レコードタイプのノードとポートの次の例を示す。これは図 7 と図 8 に示したものと同じだが、ここではエッジをつなぐポートを指定している。レコードタイプでは、その高さを小さく指定しておくと、図 11 に示すように最終的には文字の大きさによって高さが決まるため、外見が良くなることがある。指定しなければ図 14 のように、ノードの大きさはデフォルト値（.75x.5 インチ）になる。図 15 と図 16 にハッシュテーブルの図を左から右方向に描画した例を示す。

3.2 クラスタ

クラスタとは、独自の長方形の枠を持ち、独自にレイアウトされるサブグラフである。名前の最初に `cluster` が付くサブグラフがクラスタと認識される（もし `clusterrank=none` がトップレベルのグラフで指定されていれば、こういった取り扱いはなされない）。ラベル、フォント、`labelloc` 属性はトップレベルのグラフと同様に指定でき、デフォルトではクラスタのラベルはその上部に配置される。クラスタのデフォルトではラベルは左寄せであり、`labeljust="r"`を指定すると右寄せになる。`color` 属性はクラスタの枠の色を指定する。また `style="filled"`を指定するとクラスタの外枠の長方形の内部が、そのクラスタよりも前に指定された `fillcolor` で塗りつぶされる（`fillcolor` が設定されていなければ、そのクラスタの `color` 属性が使われる）。

クラスタ内でのランクの割り当てとノードの順序は、再帰的な計算で決められる。図 17 と図 19 にクラスタのレイアウトとファイルの例を示す。

トップレベルのグラフで `compound` 属性が true に設定されている場合、*dot* ではノードとクラスタを結ぶエッジを描くことができる。そのためにはエッジで `lhead` および `ltail` 属性が指定されなければならない。これらの属性値にはエッジの先端または終端のノードを持つクラスタの名前を指定する。この場合はエッジはクラスタの輪郭線でクリッピングされる。`arrowhead` や `dir` などのエッジの他の属性は、そのクリッピングされたエッジに適用される。例として図 20 に `compound` 属性を使ったグラフと、描画された図を示す。

3.3 集約

トップレベルのグラフで `concentrate=true` を設定すると、密度の高い混み合ったグラフをすっきりさせるために、エッジがまとめられる。並列で同じ端点を持ち、長さが 1 よりも大きなエッジがまとめられる。図の大きさが固定されているときは、これによってラベルが大きくなって読みやすくなることもある。*dot* が用いる集約法はニューベリーの方法 [New89] に似ているが、グラフ全体中の全ての二部グラフ（偶グラフ）を探索することなく図中のそういったエッジを探す。そのため *dot* の方法ではニューベリーのアルゴリズムよりも高速で、失敗することもより少ない。

4 コマンドライン・オプション

デフォルトでは *dot* はフィルタモードで実行され、標準入力 *stdin* からグラフを読み込み、標準出力 *stdout* に *DOT* 形式で属性付きのレイアウト図を出力する。*dot* には多くのコマンドライン・オプションがあり、*-Tformat* の形式で出力形式を指定できる。*format* には以下のものが使える。

canon 入力をそのまま出力する。何も描画されない。

dot 属性付き *DOT*。入力に属性を持ったレイアウト情報を付けて出力する。付録 C 参照。
fig FIG 形式。
gd GD 形式。GD グラフィクス・ライブラリの内部で使われている形式。`gd2` も代わりに使える。
gif GIF 形式。
hpgl HP の大型プロッタで使われている HP-GL/2 ベクトル・グラフィクス・プリンタ言語。
imap サーバーサイドのイメージ・マップに使う画像ファイルを出力する。これは`-Tgif` や`-Tjpg` などの他の形式の指定と同時に用い、ノードやエッジに web ページ上でのリンクを設定することができる。`ismap` は `imap` 形式の旧版である。
cmap クライアント・サイドの HTML マップ画像を出力する。
jpg JPEG 形式。`jpeg` も同じ。
mif FrameMaker の MIF 形式。この形式の出力は FrameMaker に読み込んで編集することができる。MIF では色数が 8 色に限られている。
mp MetaPost 形式。
pcl HP のレーザープリンタで使われる PCL-5 形式。
pic PIC 形式。
plain 単純な行指向のテキスト形式。付録 B 参照。エッジの両端のノードのポート名を出力する `plain-ext` もある。
png PNG（Portable Network Graphics）形式。
ps PostScript（EPSF）形式。
ps2 PDF アノテーション付きの PostScript（EPSF）形式。PDF に変換されることを前提とした形式。
svg SVG 形式。`svgz` で圧縮された SVG 形式も出力できる。**vrml** VRML 形式。**vtx** r Confluencts の Visual Thought で使われる VTX 形式。**wbmp** Wireless BitMap（WBMP）形式。

`-G`*name*`=`*value* でデフォルトの属性値を設定できる。図の大きさやページの割付、そのほかのこうったい属性はファイル中よりもコマンドラインで指定した方が便利なこともある。同様に`-N` や `-E` でノード、エッジの属性のデフォルト値を設定できる。コマンドラインでの指定よりも、ファイル中の記述の方が優先される。

`-l`*libfile* で出力形式によって異なるようなライブラリ・ファイルを読み混むことができる。複数指定しても良い。指定されたファイル名は出力の最初にコードジェネレータに渡される。

`-O`*outfile* は出力を *outfile* という名前のファイルにする。

`-v` でメッセージを出力させることができる。大きなグラフを処理するときは、これにより *dot* の処理がどのように進むかを見て取ることができる。

`-V` を指定するとバージョンを表示してそのまま終了する。

5　その他

トップレベルのグラフの冒頭で、`strict digraph` と宣言することもできる。これにより、両端が同じノードになるエッジと重複したエッジが使えなくなり、入力ファイル中にあるそういったエッジは無視される。

ノード、エッジ、グラフには URL 属性を付けることができる。いくつかの出力形式（`ps2`、`imap`、`ismap`、`cmap`、`svg`）ではこの属性が出力に取り込まれ、web ブラウザなどで見たときにノード、エッジ、クラスタにはリンクが設定されていることになる。トップレベルのグラフに設定された

URL 属性は base URL と解釈され、グラフ中では相対的な URL を属性値として指定できる。出力形式が `imap` または `cmap` のとき、`headURL` および `tailURL` 属性が同様に指定できる。

いくつかの出力形式（`ps`、`fig`、`mif`、`mp`、`vtx`、`svg`）では、`comment` 属性を使って出力の中に人が読めるコメントを埋め込むことができる。

6 まとめ

dot は満足できるレベルの階層グラフを描くことができ、様々な設定を反映することができる。

dot の基本的なアルゴリズムはうまく動作しており、大規模なグラフやオンライン（のアニメーションなど）のグラフ描画といった将来的な課題にも適応できるだろう。

7 備考

Phong Vo の描画アルゴリズムと実装についての助言に感謝したい。描画ライブラリでは Phong の扇形ツリー辞書ライブラリを用いている。また *dot* のプリプロセッサである *dag* のユーザーも我々に多くの示唆を与えてくれた。Guy Jacobson と Randy Hackarth はこの文書の草稿をみてくれた。Emden はその後の現バージョンの貢献してくれた。John Ellson は汎用的な多角形描画法を書き、それを安定した効率の高いものにするために多くの時間を費やしてくれた。彼はさらに GIF と ISMAP の出力ルーチンと、*graphviz* を web に利用するためのその他のツールを書いてくれた。

参考文献

[Car80] M. Carpano. Automatic display of hierarchized graphs for computer aided decision analysis. *IEEE Transactions on Software Engineering*, SE-12(4):538-546, April 1980.

[GKNV93] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem-Phong Vo. A Technique for Drawing Directed Graphs. *IEEE Trans. Sofware Eng.*, 19(3):214-230, May 1993.

[New89] Frances J. Newbery. Edge Concentration: A Method for Clustering Directed Graphs. In *2nd International Workshop on Software Configuration Management*, pages 76-85, October 1989. Published as *ACM SIGSOFT Software Engineering Notes, vol. 17, no. 7*, November 1989.

[Nor92] Stephen C. North. Neato User's Guide. Technical Report 59113921014-14TM, AT&T Bell Laboratories, Murray Hill, NJ, 1992.

[STT81] K. Sugiyama, S. Tagawa, and M. Toda. Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(2):109-125, February 1981.

[War77] John Warfield. Crossing Theory and Hierarchy Mapping. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-7(7):505-523, July 1977.

翻訳に際しての訂正

- p. 5, Figure 4. Figure 3 を処理すると `main -> printf` のエッジは赤くなるので、そうした。
- p. 7, Figure 5. 色 `blue_light` はないというエラーが出るので、代わりに `blueviolet` と変更し、それを使って図 6 を出力した。
- Figure 9. ソースリストが一部だったのを全部載せた。
- 付録のノードの形とエッジの形は、現状に会わせて増やした。
- 使える色名が非常に増えているので、それも増やした。
- 面倒なので、図と表は末尾にまとめている。

A　グラフファイルの文法

DOT 言語の文法の概要を以下に示す。終端記号は太字で、非終端記号は斜体で示す。リテラル文字はシングル・クオートでかこっている。丸括弧 (と) は必要に応じてグルーピングを表す。角括弧 [と] は省略可能な要素であることを示す。縦線 | は選択肢を列挙するのに用いている。

graph	→	[**strict**](**digraph** | **graph**)*id*'{'*stmt-list*'}'
stmt-list	→	[*stmt*[';'][*stmt-list*]]
stmt	→	*attr-stmt* | *node-stmt* | *edge-stmt* | *subgraph* | *id* '=' *id*
attr-stmt	→	(**graph** | **node** | **edge**) *attr-list*
attr-list	→	'[' [*a-list*] ']' [*attr-list*]
a-list	→	*id* '=' *id*[','] [*attr-list*]
node-stmt	→	*node-id* [*attr-list*]
node-id	→	*id*[*port*]
port	→	*port-location* [*port-angle*] | *port-angle* [*port-location*]
port-location	→	':' *id* | ':' '(' *id* ',' *id* ')'
port-angle	→	'@' *id*
edge-stmt	→	(*node-id* | *subgraph*) *edgeRHS* [*attr-list*]
edgeRHS	→	*edgeop* (*node-id* | *subgraph*) [*edgeRHS*[
subgraph	→	**subgraph** *id* '' *stmt-list* '' | **subgraph** *id*

ここで *id* は任意の英数字からなる文字列で先頭が数字でないものだが、下線、数字、クオートでくくられたエスケープ・シーケンスを含む任意の文字列を入れてもかまわない。

edgeop は有向グラフでは->、無向グラフでは--である。

この言語では C++風のコメントを使うことができる。`/* */`と`//`である。

セミコロンがあると可読性が良くなるが、ほとんどの場合はなくても良い。しかし本体がなく名前を持つサブグラフが名前のないサブグラフの直前にある場合には必要である。これは、以前の規則ではこういった記述はサブグラフのヘッダと本体であると見なされたからである。

属性値が複雑になるような場合、*DOT* 言語で構文解析に用いるコンマや空白文字が含まれることがある。構文エラーを避けるために、そういった属性値はクオーテーションでくくる必要がある。

B　テキスト出力のフォーマット

「テキスト」出力形式では、フロントエンドのプログラムで簡単に解析できるように、ノードとエッジの情報を、各行ごとに記述する。座標値は全てスケーリングされず、単位はインチである。

出力の最初の行は以下のようになる。

`graph` *scalefactor width height*

width と *height* は図の幅と高さである。図の左下の隅が座標の原点である。*scalefactor* は、図の縮尺である。

続いてノードのリストとなる行が記述される。

`node` *name x y xsize ysize label style shape color fillcolor*

name はノードの識別子である。識別子に空白文字などが含まれる場合はクオートされる。*x* と *y* はノードの中心の座標である。*xsize* と *ysize* はそれぞれ幅と高さである。他のパラメータはそれぞ

れノードの label、style、shape、color、fillcolor 属性である。ノードに style 属性が定義されてない場合は solid になる。

その次にはエッジの行が続く。

edge *tail head* n $x_1y_1x_2y_2 \ldots x_ny_n$ *[label lx ly] style color*

n はエッジの曲線を定義する B スプラインの座標の組の数である。ラベルがある場合は、ラベルの文字列と座標がそれに続く。style と color が最後に定義される。ノードと同様に、style が定義されていない場合は solid が適用される。

出力の最後の行は常に以下の行である。

stop

C 属性付き DOT フォーマット (-Tdot)

この出力形式がデフォルトである。入力にグラフのレイアウト情報を付け足したものである。座標の値は上、および右に行くにしたがって増える。位置はポイント（1/17 インチ）単位で、X および Y 座標に対応する二つの整数をコンマで区切って指定される。位置はオブジェクトの中心についてである。長さは単位インチで与えられる。

グラフに与えられた bb 属性は、図のバウンディング・ボックスを指定する。グラフにラベルが付けられている場合は、lp 属性でその位置が指定される。

各ノードには pos、width、height の各属性が与えられる。ノードがレコードタイプの場合、レコードを表す長方形は rects 属性で与えられる。ノードが多角形で、入力でその vertices 属性が指定されている場合は、それが出力中でも指定される。円や楕円を描く際にいくつの点を生成するかは、samplepoints 属性で指定される。

エッジには、$3n+1$ 組の位置座標を指定する pos 属性が与えられる。これは B スプラインの制御点である。点 p_0、p_1、p_2、p_3 が最初のベジェ・スプライン、点 p_3、p_4、p_5、p_6 が次のベジェ・スプライン、などである。現在のところ、エッジの向きにかかわらず点の順序は上から下（または左から右）である。これは変更される可能性がある。

pos 属性の定義中で、制御点のリストの前に開始点 p_s と終始点 p_e の両方または一方を指定できる。これは先頭に"s"または"e"をそれぞれ付けて普通の位置と同様に指定される。エッジの先端、p_0 に矢印がある場合には開始点を指定できる。矢印は p_0 から p_s に描かれる。p_s は実際にはノードの境界線上になる。矢印の大きさはベクトル $(p_s - p_0)$ で与えられる。矢印がない場合は p_0 はノードの境界線上になる。同様に p_e はエッジのもう一方の端について同様に扱われ、スプラインの最後の点をつなぐ。

エッジにラベルが付けられている場合、その位置は lp 属性で指定される。

D レイヤー

dot には一連の「レイヤー」を重ねて一つの図の一部を描く機能がある。レイヤーは OHP のシートのようなものである。この機能を使うには、トップレベルのグラフで識別子のリストを layer 属性で指定する。これによりノードやエッジに layer 属性にレイヤーやレイヤーの範囲などを指定することで、そのオブジェクトをレイヤーに配置できる。識別子 all は全てのレイヤーを表現す

るために予約語になっている（たとえば `design:all` や `all:code` のように、範囲指定の一方に用いることもできる）。例を挙げると、

```
layers = "spec:design:code:debug:ship";
node90 [layer = "code"];
node91 [layer = "design:debug"];
node90 -> node91 [layer = "all"];
node92 [layer = "all:code"];
```

このグラフでは、`node91` は 3 つのレイヤー `design`、`code`、`debug` に配置される。一方 `node92` はレイヤー `spec`、`design`、`code` に配置される。

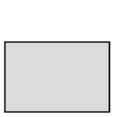
レイヤーを持つグラフでは、ノードやエッジに配置されるレイヤーが指定されておらず、しかもそれに接続するエッジやノードはレイヤーが指定されている場合、指定されていないオブジェクトもレイヤーに配置される。レイヤー指定のないオブジェクトが全てのレイヤーで見えるようにするには、グラフ定義の最初に以下の行を入れる。

```
node [layer=all];
edge [layer=all];
```

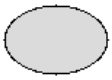
現在はまだ、連続しない複数のレイヤーを指定する方法はない。

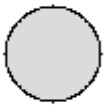
出力が PostScript 形式の場合、`layercolorseq` でレイヤーにカラーシーケンスを指定できる。これは 1 から始まる配列で、各要素は色座標を表す 3 要素の配列である必要がある。興味のある場合は *dot* の PostScript 出力を見てもらいたい。

E　ノードの形


box


polygon


ellipse

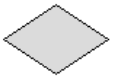

circle

point


egg

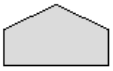

triangle

plaintext

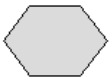

diamond


trapezium

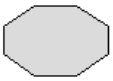

parallelogram

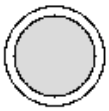

house

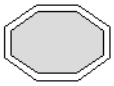

pentagon

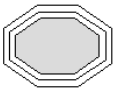

hexagon

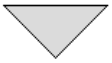

septagon


octagon

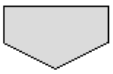

doublecircle

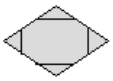

doubleoctagon

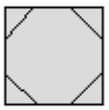

tripleoctagon

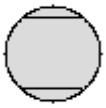

invtriangle


invtrapezium


invhouse

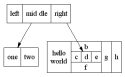

Mdiamond


Msquare

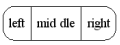

Mcircle


rect


rectangle

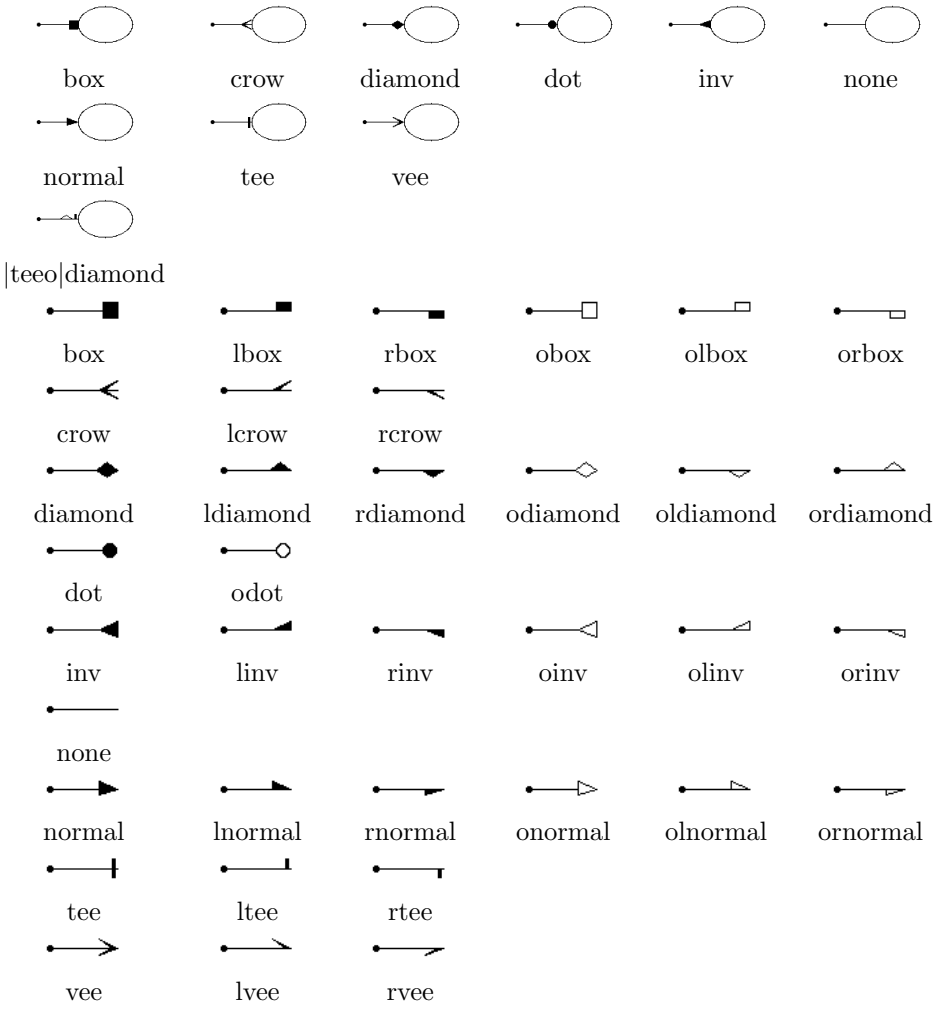


record



mrecord

F　矢印の形


box
crow
diamond
dot
inv
none
normal
tee
vee
|teeo|diamond
box
lbox
rbox
obox
olbox
orbox
crow
lcrow
rcrow
diamond
ldiamond
rdiamond
odiamond
oldiamond
ordiamond
dot
odot
inv
linv
rinv
oinv
olinv
orinv
none
normal
lnormal
rnormal
onormal
olnormal
ornormal
tee
ltee
rtee
vee
lvee
rvee

G　色名

aliceblue	antiquewhite	antiquewhite1	antiquewhite2	antiquewhite3
antiquewhite4	aquamarine	aquamarine1	aquamarine2	aquamarine3
aquamarine4	azure	azure1	azure2	azure3
azure4	beige	bisque	bisque1	bisque2
bisque3	bisque4	black	blanchedalmond	blue
blue1	blue2	blue3	blue4	blueviolet
brown	brown1	brown2	brown3	brown4
burlywood	burlywood1	burlywood2	burlywood3	burlywood4
cadetblue	cadetblue1	cadetblue2	cadetblue3	cadetblue4
chartreuse	chartreuse1	chartreuse2	chartreuse3	chartreuse4
chocolate	chocolate1	chocolate2	chocolate3	chocolate4
coral	coral1	coral2	coral3	coral4
cornflowerblue	cornsilk	cornsilk1	cornsilk2	cornsilk3
cornsilk4	crimson	cyan	cyan1	cyan2
cyan3	cyan4	darkgoldenrod	darkgoldenrod1	darkgoldenrod2
darkgoldenrod3	darkgoldenrod4	darkgreen	darkkhaki	darkolivegreen
darkolivegreen1	darkolivegreen2	darkolivegreen3	darkolivegreen4	darkorange
darkorange1	darkorange2	darkorange3	darkorange4	darkorchid
darkorchid1	darkorchid2	darkorchid3	darkorchid4	darksalmon
darkseagreen	darkseagreen1	darkseagreen2	darkseagreen3	darkseagreen4
darkslateblue	darkslategray	darkslategray1	darkslategray2	darkslategray3
darkslategray4	darkslategrey	darkturquoise	darkviolet	deeppink
deeppink1	deeppink2	deeppink3	deeppink4	deepskyblue
deepskyblue1	deepskyblue2	deepskyblue3	deepskyblue4	dimgray
dimgrey	dodgerblue	dodgerblue1	dodgerblue2	dodgerblue3
dodgerblue4	firebrick	firebrick1	firebrick2	firebrick3
firebrick4	floralwhite	forestgreen	gainsboro	ghostwhite
gold	gold1	gold2	gold3	gold4
goldenrod	goldenrod1	goldenrod2	goldenrod3	goldenrod4
gray	gray0	gray1	gray10	gray100
gray11	gray12	gray13	gray14	gray15
gray16	gray17	gray18	gray19	gray2
gray20	gray21	gray22	gray23	gray24
gray25	gray26	gray27	gray28	gray29
gray3	gray30	gray31	gray32	gray33
gray34	gray35	gray36	gray37	gray38
gray39	gray4	gray40	gray41	gray42
gray43	gray44	gray45	gray46	gray47
gray48	gray49	gray5	gray50	gray51
gray52	gray53	gray54	gray55	gray56
gray57	gray58	gray59	gray6	gray60

gray61	gray62	gray63	gray64	gray65
gray66	gray67	gray68	gray69	gray7
gray70	gray71	gray72	gray73	gray74
gray75	gray76	gray77	gray78	gray79
gray8	gray80	gray81	gray82	gray83
gray84	gray85	gray86	gray87	gray88
gray89	gray9	gray90	gray91	gray92
gray93	gray94	gray95	gray96	gray97
gray98	gray99	green	green1	green2
green3	green4	greenyellow	grey	grey0
grey1	grey10	grey100	grey11	grey12
grey13	grey14	grey15	grey16	grey17
grey18	grey19	grey2	grey20	grey21
grey22	grey23	grey24	grey25	grey26
grey27	grey28	grey29	grey3	grey30
grey31	grey32	grey33	grey34	grey35
grey36	grey37	grey38	grey39	grey4
grey40	grey41	grey42	grey43	grey44
grey45	grey46	grey47	grey48	grey49
grey5	grey50	grey51	grey52	grey53
grey54	grey55	grey56	grey57	grey58
grey59	grey6	grey60	grey61	grey62
grey63	grey64	grey65	grey66	grey67
grey68	grey69	grey7	grey70	grey71
grey72	grey73	grey74	grey75	grey76
grey77	grey78	grey79	grey8	grey80
grey81	grey82	grey83	grey84	grey85
grey86	grey87	grey88	grey89	grey9
grey90	grey91	grey92	grey93	grey94
grey95	grey96	grey97	grey98	grey99
honeydew	honeydew1	honeydew2	honeydew3	honeydew4
hotpink	hotpink1	hotpink2	hotpink3	hotpink4
indianred	indianred1	indianred2	indianred3	indianred4
indigo	ivory	ivory1	ivory2	ivory3
ivory4	khaki	khaki1	khaki2	khaki3
khaki4	lavender	lavenderblush	lavenderblush1	lavenderblush2
lavenderblush3	lavenderblush4	lawngreen	lemonchiffon	lemonchiffon1
lemonchiffon2	lemonchiffon3	lemonchiffon4	lightblue	lightblue1
lightblue2	lightblue3	lightblue4	lightcoral	lightcyan
lightcyan1	lightcyan2	lightcyan3	lightcyan4	lightgoldenrod
lightgoldenrod1	lightgoldenrod2	lightgoldenrod3	lightgoldenrod4	lightgoldenrodyellow
lightgray	lightgrey	lightpink	lightpink1	lightpink2
lightpink3	lightpink4	lightsalmon	lightsalmon1	lightsalmon2

lightsalmon3	lightsalmon4	lightseagreen	lightskyblue	lightskyblue1
lightskyblue2	lightskyblue3	lightskyblue4	lightslateblue	lightslategray
lightslategrey	lightsteelblue	lightsteelblue1	lightsteelblue2	lightsteelblue3
lightsteelblue4	lightyellow	lightyellow1	lightyellow2	lightyellow3
lightyellow4	limegreen	linen	magenta	magenta1
magenta2	magenta3	magenta4	maroon	maroon1
maroon2	maroon3	maroon4	mediumaquamarine	mediumblue
mediumorchid	mediumorchid1	mediumorchid2	mediumorchid3	mediumorchid4
mediumpurple	mediumpurple1	mediumpurple2	mediumpurple3	mediumpurple4
mediumseagreen	mediumslateblue	mediumspringgreen	mediumturquoise	mediumvioletred
midnightblue	mintcream	mistyrose	mistyrose1	mistyrose2
mistyrose3	mistyrose4	moccasin	navajowhite	navajowhite1
navajowhite2	navajowhite3	navajowhite4	navy	navyblue
oldlace	olivedrab	olivedrab1	olivedrab2	olivedrab3
olivedrab4	orange	orange1	orange2	orange3
orange4	orangered	orangered1	orangered2	orangered3
orangered4	orchid	orchid1	orchid2	orchid3
orchid4	palegoldenrod	palegreen	palegreen1	palegreen2
palegreen3	palegreen4	paleturquoise	paleturquoise1	paleturquoise2
paleturquoise3	paleturquoise4	palevioletred	palevioletred1	palevioletred2
palevioletred3	palevioletred4	papayawhip	peachpuff	peachpuff1
peachpuff2	peachpuff3	peachpuff4	peru	pink
pink1	pink2	pink3	pink4	plum
plum1	plum2	plum3	plum4	powderblue
purple	purple1	purple2	purple3	purple4
red	red1	red2	red3	red4
rosybrown	rosybrown1	rosybrown2	rosybrown3	rosybrown4
royalblue	royalblue1	royalblue2	royalblue3	royalblue4
saddlebrown	salmon	salmon1	salmon2	salmon3
salmon4	sandybrown	seagreen	seagreen1	seagreen2
seagreen3	seagreen4	seashell	seashell1	seashell2
seashell3	seashell4	sienna	sienna1	sienna2
sienna3	sienna4	skyblue	skyblue1	skyblue2
skyblue3	skyblue4	slateblue	slateblue1	slateblue2
slateblue3	slateblue4	slategray	slategray1	slategray2
slategray3	slategray4	slategrey	snow	snow1
snow2	snow3	snow4	springgreen	springgreen1
springgreen2	springgreen3	springgreen4	steelblue	steelblue1
steelblue2	steelblue3	steelblue4	tan	tan1
tan2	tan3	tan4	thistle	thistle1
thistle2	thistle3	thistle4	tomato	tomato1
tomato2	tomato3	tomato4	transparent	turquoise
turquoise1	turquoise2	turquoise3	turquoise4	violet

```
1: digraph G {
2:     main -> parse -> execute;
3:     main -> init;
4:     main -> cleanup;
5:     execute -> make_string;
5:     execute -> printf;
6:     init -> make_string;
8:     main -> printf;
9:     execute -> compare;
10: }
```

図 1: Small graph

violetred	violetred1	violetred2	violetred3	violetred4
wheat	wheat1	wheat2	wheat3	wheat4
white	whitesmoke	yellow	yellow1	yellow2
yellow3	yellow4	yellowgreen		

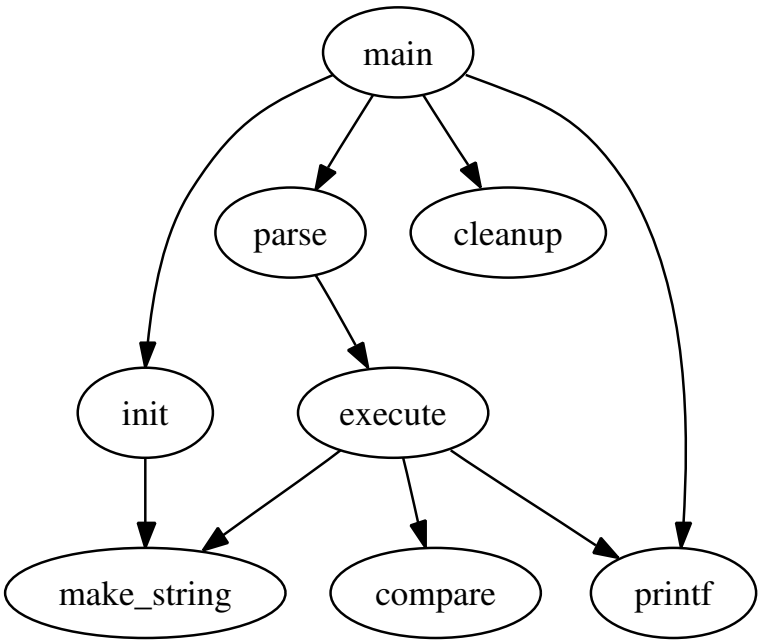


図 2: Drawing of small graph

```
1:  digraph G {
2:      size = "4, 4";
3:      main [shape=box];   /* コメントも書けます */
4:      main -> parse [weight=8];
5:      parse -> execute;
6:      main -> init [style=dotted];
7:      main -> cleanup;
8:      execute -> { make_string; printf}
9:      init -> make_string;
10:     edge [color=red];  // ここもコメント
11:     main -> printf [style=bold, label="100 times"];
12:     make_string [label="make a\nstring"];
13:     node [shape=box, style=filled, color=".7 .3 1.0"];
14:     execute -> compare;
15: }
```

図 3: Fancy graph

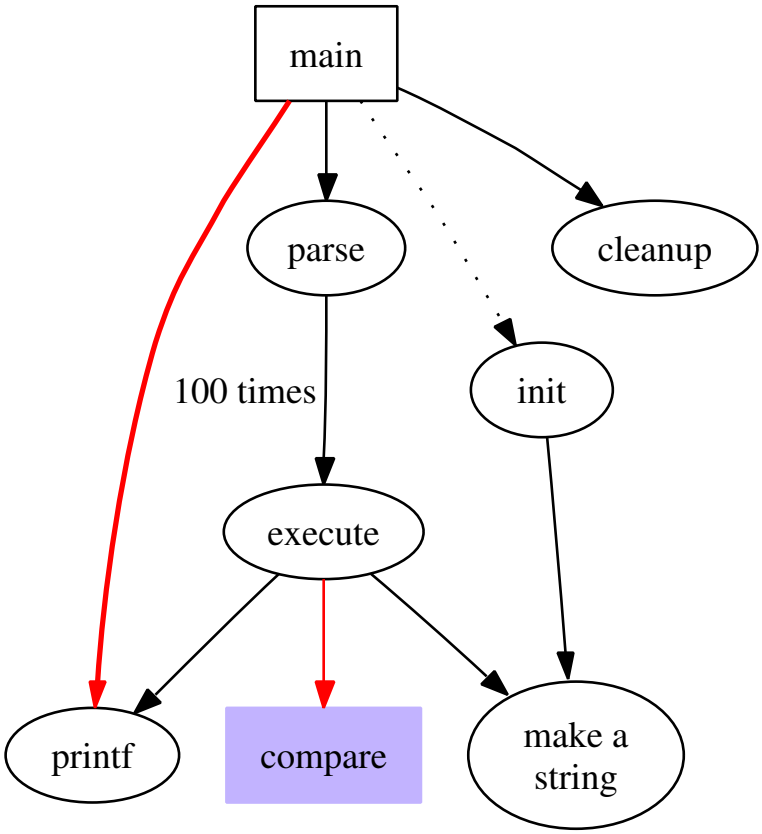


図 4: Drawing of fancy graph

```
1:  digraph G {
2:      a -> b -> c;
3:      b -> d;
4:      a [shape=polygon,sides=5,peripheries=3,color=blue_light,style=filled];
5:      c [shape=polygon,sides=4,skew=.4,label="hello world"];
6:      d [shape=invtriangle];
7:      e [shape=polygon,sides=4,distortion=.7];
8:  }
```

図 5: Graph with polygonal shapes

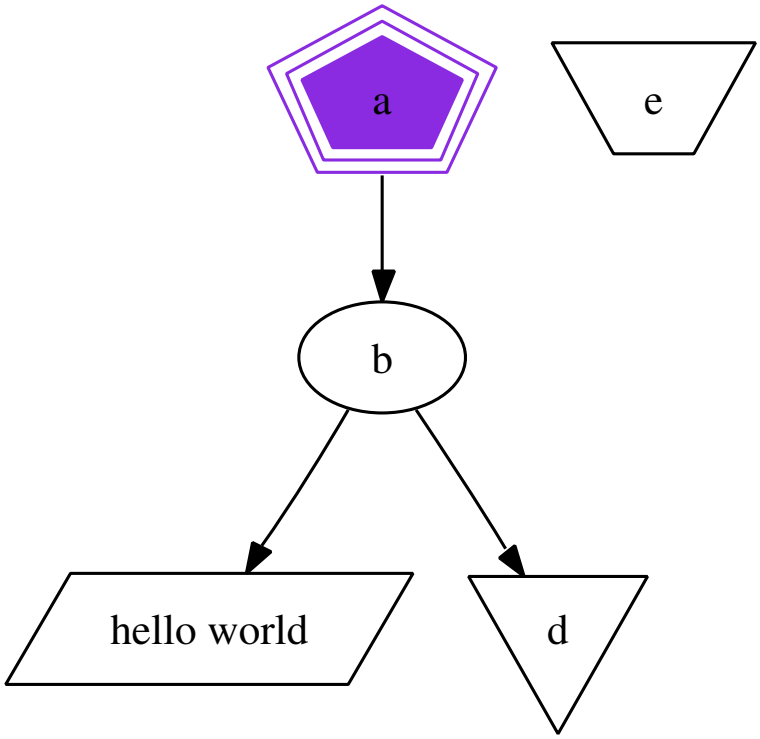


図 6: Drawing of polygonal node shapes

```
1:  digraph structs {
2:      node [shape=record];
3:      struct1 [shape=record,label="<f0> left|<f1> mid\ dle|<f2> right"];
4:      struct2 [shape=record,label="<f0> one|<f1> two"];
5:      struct3 [shape=record,label="hello\nworld|{b|{c|<here> d|e}|f}|g|h"];
6:      struct1 -> struct2;
7:      struct1 -> struct3;
8:  }
```

図 7: Record with nested fields

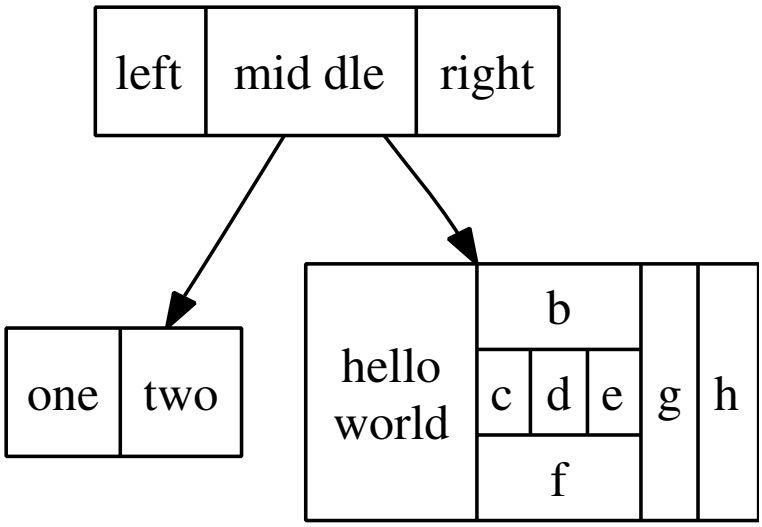


図 8: Drawing of records

```
digraph asde91 {
    ranksep=.75; size="7.5,7.5";
    {    node [shape=plaintext, fontsize=16];
        /* 時間軸のグラフ */
        past -> 1978 -> 1980 -> 1982 -> 1983 -> 1985 -> 1986 ->
                1987 -> 1988 -> 1989 -> 1990 -> "future";
        /* 系図 */
        "Bourne sh"; "make"; "SCCS"; "yacc"; "cron"; "Reiser cpp";
        "Cshell"; "emacs"; "build"; "vi"; "<curses>"; "RCS"; "C*";
    }
    {   rank = same;
        "Software IS"; "Configuration Mgt"; "Architecture & Libraries"; "Process";
    };

    node [shape=box];
    { rank=same; "past"; "SCCS"; "make"; "Bourne sh"; "yacc"; "cron"; }
    { rank=same; 1978; "Reiser cpp"; "Cshell"; }
    { rank=same; 1980; "build"; "emacs"; "vi"; }
    { rank=same; 1982; "RCS"; "<curses>"; "IMX"; "SYNED"; }
    { rank=same; 1983; "ksh"; "IFS"; "TTU"; }
    { rank=same; 1985; "nmake"; "Peggy"; }
    { rank=same; 1986; "C*"; "ncpp"; "ksh-i"; "<curses-i>"; "PG2"; }
    { rank=same; 1987; "Ansi cpp"; "nmake 2.0"; "3D File System"; "fdelta"; "DAG"; "CSAS"; }
    { rank=same; 1988; "CIA"; "SBCS"; "ksh-88"; "PEGASUS/PML"; "PAX"; "backtalk"; }
    { rank=same; 1989; "CIA++"; "APP"; "SHIP"; "DataShare"; "ryacc"; "Mosaic"; }
    { rank=same; 1990; "libft";"CoShell";"DIA";"IFS-i";"kyacc";"sfio";"yeast";"ML-X";"DOT"; }
    { rank=same; "futrue"; "Adv. Software Technology"; }

    "PEGASUS/PML" -> "ML-X";            "SCCS"      -> "nmake"; "SCCS"  -> "3D File System";
    "make"        -> "nmake";           "make"      -> "build";         "SCCS"      -> "RCS";
    "Bourne sh"   -> "Cshell";          "Bourne sh" -> "ksh";           "yacc"      -> "ryacc";
    "cron"        -> "yeast";           "Cshell"    -> "ksh";           "Reiser cpp"-> "ncpp";
    "build"       -> "nmake 2.0";       "vi"        -> "<curses>";      "vi"        -> "ksh";
    "emacs"       -> "ksh";             "RCS"       -> "fdelta";        "<curses>"-> "<curses-i>";
    "SYNED"       -> "Peggy";           "IMX"       -> "TTU";           "ksh"       -> "nmake";
     "ksh"        -> "ksh-i";           "ksh"       -> "ksh-88";        "IFS"     -> "<curses-i>";
    "IFS"         -> "sfio";            "IFS"       -> "IFS-i";         "TTU"       -> "PG2";
    "nmake"       -> "ncpp";            "nmake"     -> "3D File System";"nmake"   -> "nmake 2.0";
    "Peggy"       -> "PEGASUS/PML";     "Peggy"     -> "ryacc";         "C*"        -> "CSAS";
    "ncpp"        -> "Ansi cpp";        "<curses-i>"-> "fdelta";        "ksh-i"     -> "ksh-88";
    "PG2"         -> "backtalk";        "DAG"       -> "Software IS";   "DAG"       -> "DOT";
    "DAG"         -> "DIA";             "CSAS"      -> "CIA";           "fdelta"    -> "SBCS";
    "fdelta"      -> "PAX";             "Ansi cpp"   -> "Configuration Mgt";
    "CIA"         -> "CIA++";           "CIA"       -> "DIA";
    "nmake 2.0" -> "CoShell";           "PAX"       -> "SHIP";          "ksh-88"      -> "sfio";
    "nmake 2.0"   -> "Configuration Mgt";           "SBCS"        -> "Configuration Mgt";
    "ksh-88"      -> "Configuration Mgt";           "ksh-88"      -> "Architecture & Libraries";
    "PEGASUS/PML" -> "Architecture & Libraries";
    "PEGASUS/PML" -> "ML-X";            "backtalk"  -> "DataShare";     "APP"         -> "DIA";
    "CIA++"       -> "Software IS";     "APP"       -> "Software IS";   "ryacc"       -> "kyacc";
    "Mosaic"      -> "Processs";        "SHIP"      -> "Configuration Mgt";
    "DOT"         -> "Software IS";     "DIA"       -> "Software IS";
    "libft"       -> "Software IS";     "CoShell"   -> "Configuration Mgt";
    "yeast"       -> "Processs";        "DataShare" -> "Architecture & Libraries";
    "CoShell"     -> "Architecture & Libraries";    "sfio"        -> "Architecture & Libraries";
    "ISF-i"       -> "Architecture & Libraries";    "ML-X"        -> "Architecture & Libraries";
    "kyacc"       -> "Architecture & Libraries";"   "Software IS" -> "Adv. Software Technology";
    "Configuration Mgt"        -> "Adv. Software Technology";
    "Architecture & Libraries" -> "Adv. Software Technology";
}
```

図 9: Graph with constrained ranks

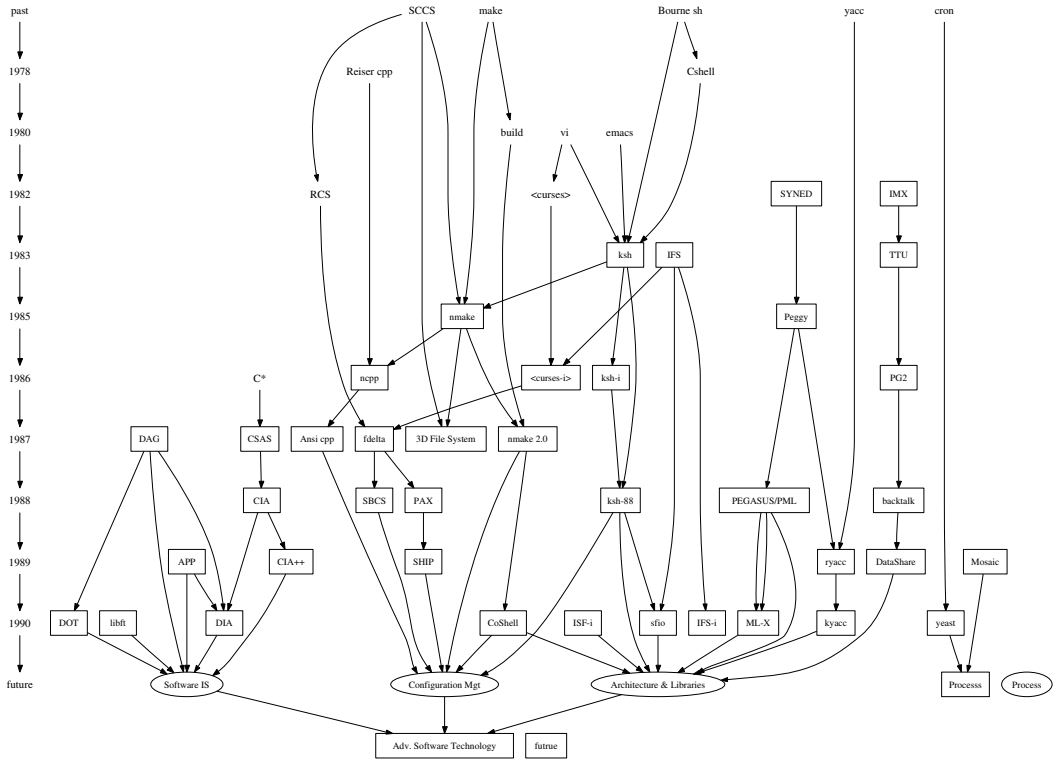


図 10: Drawing with constrained ranks

```
digraph g {
node [shape = record,height=.1];
node0[label = "<f0> |<f1> G|<f2> "];
node1[label = "<f0> |<f1> E|<f2> "];
node2[label = "<f0> |<f1> B|<f2> "];
node3[label = "<f0> |<f1> F|<f2> "];
node4[label = "<f0> |<f1> R|<f2> "];
node5[label = "<f0> |<f1> H|<f2> "];
node6[label = "<f0> |<f1> Y|<f2> "];
node7[label = "<f0> |<f1> A|<f2> "];
node8[label = "<f0> |<f1> C|<f2> "];
"node0":f2 -> "node4":f1;
"node0":f0 -> "node1":f1;
"node1":f0 -> "node2":f1;
"node1":f2 -> "node3":f1;
"node2":f2 -> "node8":f1;
"node2":f0 -> "node7":f1;
"node4":f2 -> "node6":f1;
"node4":f0 -> "node5":f1;
}
```

図 11: Binary search tree using records

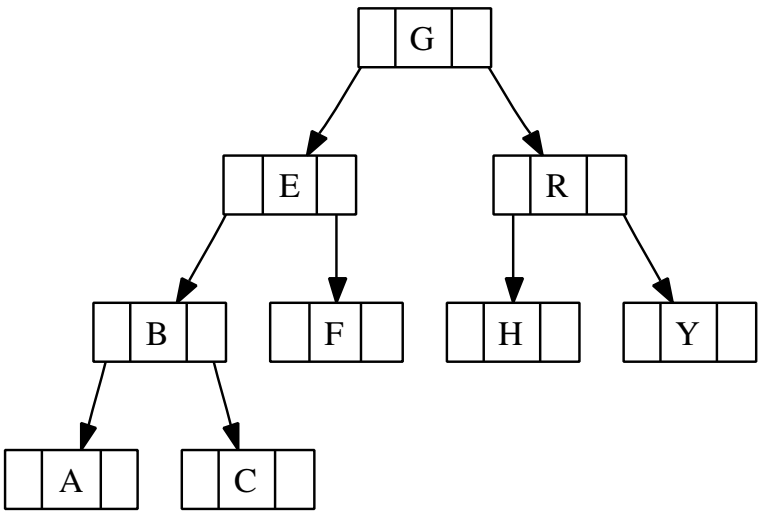


図 12: Drawing of binary search tree

```
digraph structs {
node [shape=record];
    struct1 [shape=record,label="<f0> left|<f1> middle|<f2> right"];
    struct2 [shape=record,label="<f0> one|<f1> two"];
    struct3 [shape=record,label="hello\nworld|{b|{c|<here> d|e}|f}|g|h"];
    struct1:f1 -> struct2:f0;
    struct1:f2 -> struct3:here;
}
```

図 13: Records with nested fileds (revited)

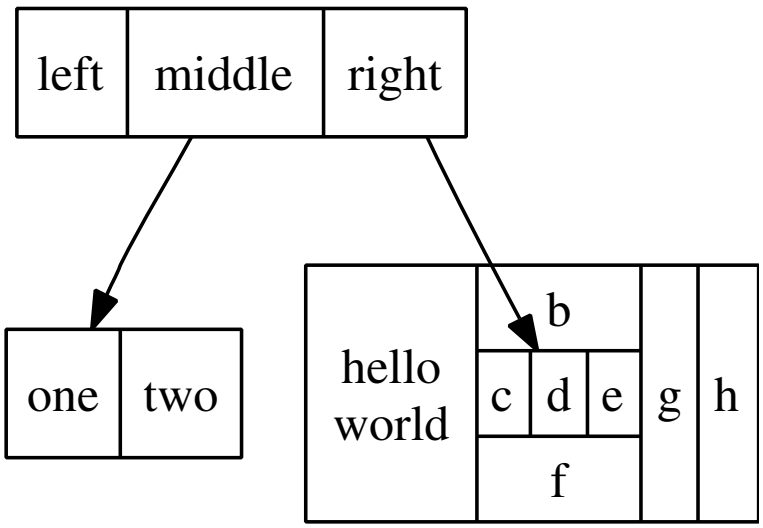


図 14: Drawing of records (revisited)

```
digraph G {
    nodesep=.05;
    rankdir=LR;
    node [shape=record,width=.1,height=.1];

    node0 [label = "<f0> |<f1> |<f2> |<f3> |<f4> |<f5> |<f6> | ",height=2.5];
    node [width = 1.5];
    node1 [label = "{<n> n14 | 719 |<p> }"];
    node2 [label = "{<n> a1  | 805 |<p> }"];
    node3 [label = "{<n> i9  | 718 |<p> }"];
    node4 [label = "{<n> e5  | 989 |<p> }"];
    node5 [label = "{<n> t20 | 959 |<p> }"];
    node6 [label = "{<n> o15 | 794 |<p> }"];
    node7 [label = "{<n> s19 | 659 |<p> }"];

    node0:f0 -> node1:n;
    node0:f1 -> node2:n;
    node0:f2 -> node3:n;
    node0:f5 -> node4:n;
    node0:f6 -> node5:n;
    node2:p -> node6:n;
    node4:p -> node7:n;
}
```

図 15: Hash table graph file

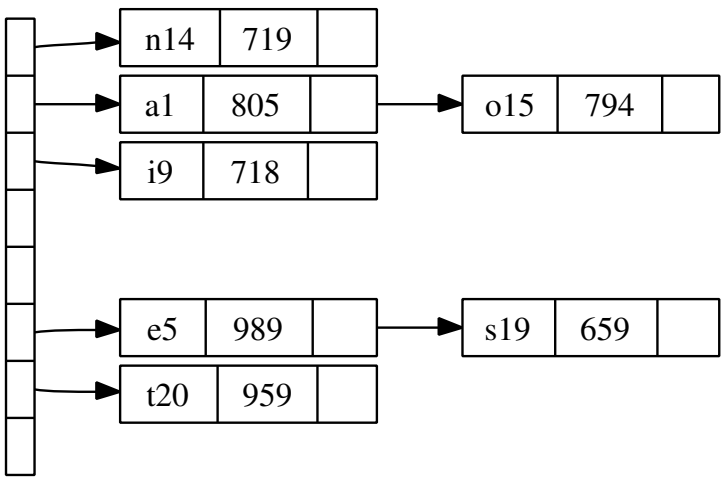


図 16: Drawing of hash table

```
digraph G {
    subgraph cluster0 {
        node [style=filled,color=white];
        style=filled;
        color=lightgrey;
        a0 -> a1 -> a2 -> a3;
        label = "process #1";
    }

    subgraph cluster1 {
        node [style=filled];
        b0 -> b1 -> b2 -> b3;
        label = "process #2";
        color=blue
    }
    start -> a0;
    start -> b0;
    a1 -> b3;
    b2 -> a3;
    a3 -> a0;
    a3 -> end;
    b3 -> end;
    start [shape=Mdiamond];
    end [shape=Msquare];
}
```

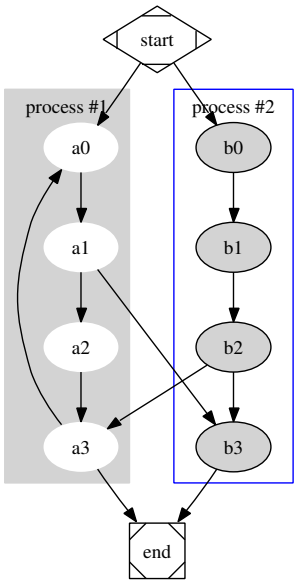


図 17: Process diagram with clusters

```
digraph G {
    size="8,6"; ratio=fill; node[fontsize=24];

    ciafan->computefan; fan->increment; computefan->fan; stringdup->fatal;
    main->exit; main->interp_err; main->ciafan; main->fatal; main->malloc;
    main->strcpy; main->getopt; main->init_index; main->strlen; fan->fatal;
    fan->ref; fan->interp_err; ciafan->def; fan->free; computefan->stdprintf;
    computefan->get_sym_fields; fan->exit; fan->malloc; increment->strcmp;
    computefan->malloc; fan->stdsprintf; fan->strlen; computefan->strcmp;
    computefan->realloc; computefan->strlen; debug->sfprintf; debug->strcat;
    stringdup->malloc; fatal->sfprintf; stringdup->strcpy; stringdup->strlen;
    fatal->exit;

    subgraph "cluster_error.h" { label="error.h"; interp_err; }
    subgraph "cluster_sfio.h" { label="sfio.h"; sfprintf; }
    subgraph "cluster_ciafan.c" { label="ciafan.c"; ciafan; computefan;
        increment; }
    subgraph "cluster_util.c" { label="util.c"; stringdup; fatal; debug; }
    subgraph "cluster_query.h" { label="query.h"; ref; def; }
    subgraph "cluster_field.h" { get_sym_fields; }
    subgraph "cluster_stdio.h" { label="stdio.h"; stdprintf; stdsprintf; }
    subgraph "cluster_<libc.a>" { getopt; }
    subgraph "cluster_stdlib.h" { label="stdlib.h"; exit; malloc; free; realloc; }
    subgraph "cluster_main.c" { main; }
    subgraph "cluster_index.h" { init_index; }
    subgraph "cluster_string.h" { label="string.h"; strcpy; strlen; strcmp; strcat; }
}
```

図 18: Call graph file

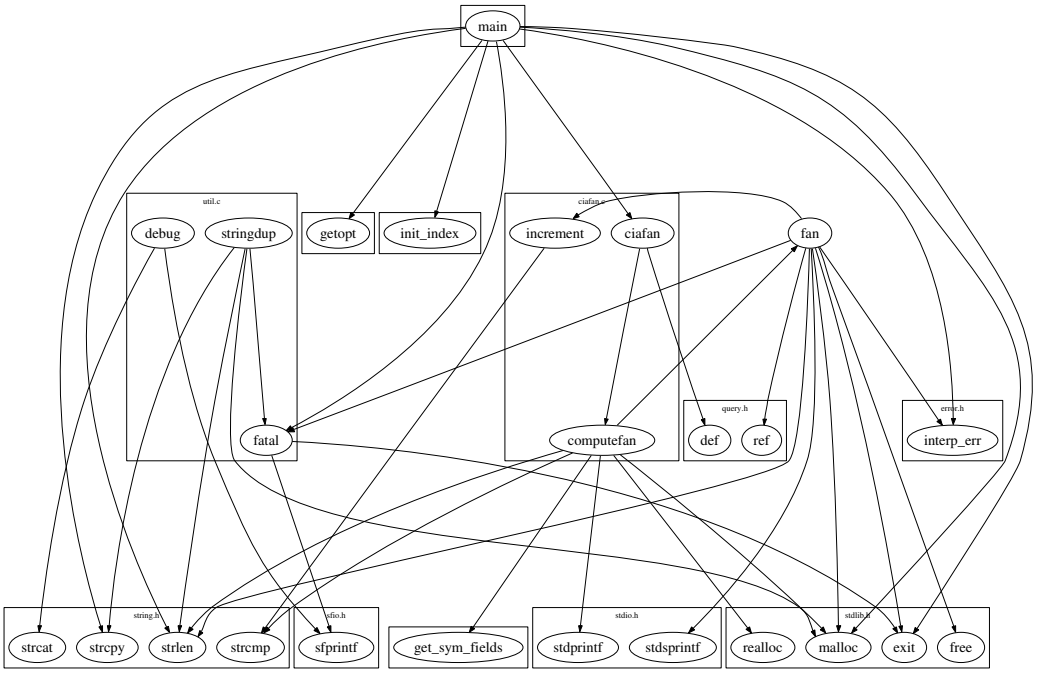


図 19: Call graph with labeled clusters

```
digraph G { compound=true;
    subgraph cluster0 {
        a -> b;
        a -> c;
        b -> d;
              c -> d;
    }
    subgraph cluster1 {
        e -> g;
        e -> f;
    }
    b -> f [lhead=cluster1];
    d -> e;
    c -> g [ltail=cluster0,
            lhead=cluster1];
    c -> e [ltail=cluster0];
    d -> h;
}
```

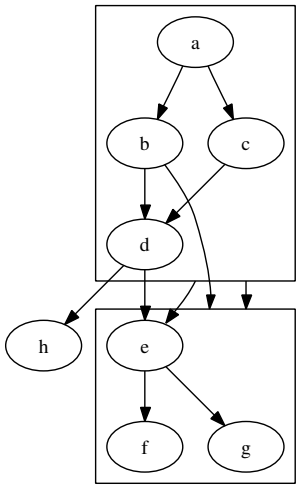


図 20: Graph with edges on clusters

Name	Default	Values
bottomlabel		shape M*での追加ラベル
color	black	ノードの輪郭の色
comment		何らかの文字列
distortion	0.0	shape=polygon でのノードのゆがみ
fillcolor	lightgrey/black	ノードを塗りつぶす色
fixedsize	false	ノードのラベル文字列がノードのサイズに影響しない
fontcolor	black	文字の色
fontname	Times-Roman	フォントファミリ
fontsize	14	ラベルの文字の大きさ
group		ノードのグループ名
height	.5	高さ（インチ）
label	ノード名	何らかの文字列
layer	overlay range	all または *id* または *id:id*
orientation	0.0	ノードを傾ける角度
peripehries	形によって異なる	ノードの輪郭線の重なる本数
regular	false	正多角形にする
shape	ellipse	ノードの形、2.1 章と付録 E 参照
shapefile		外部 EPSF または SVG ファイル
sides	4	shape=polygon での辺の数
skew	0.0	shape=polygon でのゆがみ
style		bold、dotted、filled などのオプション、2.3 章参照
toplabel		shape M*での追加ラベル
URL		ノードに付けられた URL（出力形式に依存）
width	.75	幅（インチ）
z	0.0	VRML 出力での z 座標

表 1: ノードの属性

Name	Default	Values
arrowhead	normal	先端での矢印のスタイル
arrowsize	1.0	矢印の拡大・縮小の倍数
arrowtail	normal	終端での矢印のスタイル
color	black	エッジを描く色
comment		何らかの文字列（出力形式に依存）
constraint	true	エッジがノードのランクに影響を与える
decorate		設定されていれば、ラベルとエッジの間に線が引かれる
dir	forward	forward、back、both、none
fontcolor	black	文字の色
fontname	Times-Roman	フォントファミリ
fontsize	14	ラベルのフォントの大きさ
headlabel		エッジの先端付近のラベル
headport		n,ne,e,se,s,sw,w,nw
headURL		出力が ismap のとき先端のラベルに付ける URL
label		何らかの文字列
labelangle	-25.0	両端のラベルのエッジとの角度
labeldistance	1.0	両端のラベルのノードからの距離
labelfloat	false	ラベルの位置の制約を減らす
labelfontcolor	black	両端のラベルの文字の色
labelfontname	Times-Roman	両端のラベルのフォントファミリ
labelfontsize	14	両端のラベルの文字の大きさ
layer	overlay range	all,*id*,*id:id*
lhead		エッジ先端のクラスタの名前
ltail		エッジ終端のクラスタの名前
minlen	1	エッジ両端間のランク間距離の最小値
samehead		先端のノードに付けるタグ、先端が同じタグのエッジは一つのポートにまとめられる
sametail		終端のノードに付けるタグ、終端が同じタグのエッジは一つのポートにまとめられる
style		外見を指定するオプション、2.3 章参照
taillabel		エッジ終端付近のラベル
tailport		n,ne,e,se,s,sw,w,nw
tailURL		出力が ismap のとき終端のラベルに付ける URL
weight	1	エッジを引き延ばすときの負荷

表 2: Edge attributes

Name	Default	Values
bgcolor		図の背景色および塗りつぶしの色の初期値
center	false	page の中心に図を置く
clusterrank	local	global または none
color	black	クラスタの外枠と、fillcolor がないときの塗りつぶしの色
comment		何らかの文字列（出力形式に依存）
compound	false	クラスタ間のエッジを可能にする
concentrate	false	エッジの集約を可能にする
fillcolor	black	クラスタを塗りつぶす色
fontcolor	black	文字の色
fontname	Times-Roman	フォントファミリ
fontpath		フォントのあるディレクトリのリスト
fontsize	14	ラベルの文字の大きさ
label		何らかの文字列
labeljust	centered	クラスタのラベルを左（”l”）または右（”r”）に付ける
labelloc	top	クラスタのラベルを上（”t”）または下（”b”）に付ける
layers		*id:id:id...*
margin	.5	page 内の余白（インチ）
mclimit	1.0	mincross 反復の係数
nodesep	.25	ノード間の距離（インチ）
nsmimit		f に設定すると x 座標を決めるときに network simplex の反復回数を (f)(ノード数) にする
nsmimit1		f に設定するとノードのランクを決めるときに network simplex の反復回数を (f)(ノード数) にする
ordering		out なら出るエッジの順序を保存する
orientaion	portrait	rotate が設定されず、これが landscape なら、ランドスケープの向きになる
page		ページ分割の単位、たとえば"8.5,11"
pagedir	BL	ページの出力順序
quantum		正の値ならノードラベルの向きが quantum の整数倍の角度だけ回転する
rank		same,min,max,source,sink
rankdir	TB	LR（左から右）または TB（上から下）
ranksep	.75	ランク間の距離（インチ）
ratio		おおよその縦横比、fill または auto
remincross		true でクラスタが複数あれば、交点最小化をもう一度行う
rotate		90 ならランドスケープ方向にする
samplepoints	8	出力時に円、楕円を表現するための点の数の最大値（付録 C 参照）
searchsize	30	network simplex で最小エッジを探すときの、負のカット値を持つエッジの最大数
size		図の大きさの上限
style		クラスタの filled のようなオプション
URL		グラフに設定する URL（出力形式に依存）

表 3: グラフの属性