ビタビアルゴリズムの汎用ツールの作成

執筆者：高橋 篤史

指導教官：新納 浩幸

平成13年3月2日

目 次

第1章　序論 1

1.1 背景 1

1.2 本論文の構成 1

第2章　動的計画法 2

2.1 概要 2

2.2 各種問題例とその一般的解法 4

2.2.1 割当問題 4

2.2.2 巡回セールスマン問題 7

2.2.3 貨物輸送問題（ナップサック問題） 8

第3章　ビタビアルゴリズム 11

3.1 自然言語処理とのかかわり 11

3.2 問題の設定 11

3.3 アルゴリズム 13

第4章　Javaによるビタビアルゴリズムの設計 14

4.1 Java言語 14

4.1.1 Javaについて 14

4.1.2 Java言語の特徴 15

4.1.3 オブジェクト指向 16

4.2 Javaを用いる利点 20

第5章 種々の動的計画問題への適用 21

5.1 形態素グラフの最適パスを求める問題 21

5.1.1 問題例 21

5.1.2 プログラムの構成 23

5.1.3 データファイルの表現 23

5.2 その他の動的計画法への拡張 25

5.2.1 割当問題 25

5.2.2 巡回セールスマン問題 27

5.2.3 貨物輸送問題（ナップサック問題） 30

5.3 グラフノードのデータ構造 31

第6章 考察 34

第7章 おわりに 35

付録A プログラムリスト（C言語） 38

付録B プログラムリスト（Java） 45

図目次

2.1 多段階決定過程 3
2.2 ヒッチコック型輸送問題 5
2.3 輸送問題 6

3.1 グラフの例 12

4.1 クラス継承階層 18
4.2 ポリモーフィズム 19

5.1 ビタビアルゴリズムによるコスト計算 22
5.2 単純なリストリンクの例 24
5.3 割当問題のグラフ表現 26
5.4 巡回セールスマン問題のグラフ表現 29
5.5 ナップサック問題のグラフ表現 32
5.6 データ構造 33

表目次

5.1 データの形式 25
5.2 割当問題におけるデータの形式 27
5.3 巡回セールスマン問題におけるデータの形式 30
5.4 ナップサック問題におけるデータの形式 31

第1章　序論

1.1　背景

本研究室ではこれまで，動的計画問題を解くアルゴリズムとしてビタビアルゴリズムが用いられてきた．しかし，そのプログラムはその都度作成していたため手間と労力を必要としてきた．そこで，本研究ではそのわずらわしさを解消するため，今後の研究に有用なビタビアルゴリズムの汎用的なツールを作成する．Java言語を用いてオブジェクト指向的に作成することにより汎用性を高め，汎用性を確認するために種々の動的計画問題 [1] を解く．

1.2　本論文の構成

本論文では，まず代表的な3つの動的計画問題を紹介し，それらの一般的解法について述べる（第2章）．次に，本研究で用いた動的計画法の一種であるビタビアルゴリズムについての説明を行なった後（第3章），ビタビアルゴリズムによる種々の動的計画問題への適用として，形態素グラフの最適パスを求める問題及び，その他の動的計画問題におけるグラフやデータファイルの表現方法について説明し（第4章），考察（第5章）へと進む．

また，巻末にはプログラムリストを添付した．

第2章　動的計画法

2.1　概要

動的計画法はある目的関数の最大，あるいは最小を求める最適化問題に対する手法である [2]．特に，最適化問題のなかで繰り返し部分を発見し，最適性の原理から作られる漸化式を解く方法を動的計画法と呼ぶ．また，この動的計画法で解くことができる問題を動的計画問題と呼ぶ．

動的計画法は多段決定過程を対象とする数学理論である．すなわち，その過程はいくつかの段階より成り，各段階において決定がなされなくてはならない．そして各段階における決定が，ある段階でなされるべき決定を制約する．

多段決定過程は2種類考えられる．1つめは時間的な多段決定過程で，これが最も一般的である．図2.1に示すように，矢印の向きに時間が経過し，正方形は連結した段階を示している．各段階で可能な決定群より適当な決定が選択される．

そのとき，前段階における決定がつぎの段階における決定へつぎつぎに影響していく．したがって，全段階後の目的関数の最適化のためには，各段階における決定は，以後の段階への影響を考慮してなされなければならない．

2つめは，空間的な多段決定過程である．たとえば組織の最適化などの場合には，各部門は互いに関連し，1つの部門ごとの最適化は，組織全体の最適化になるとは限らない．この場合，時間的変化はないが，空間的な意味での多段決定と考えることができる．

以上のどの意味での多段決定過程でも，部分的最適は必ずしも全体な最適に導くとは限らないことに注目しなければならない．

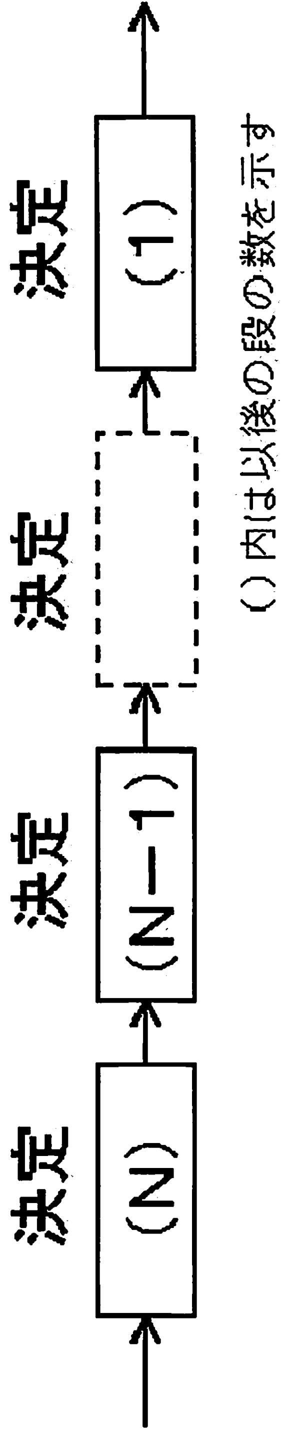

図 2.1: 多段階決定過程

2.2 各種問題例とその一般的解法

2.2.1 割当問題

n 人の社員 $\{P_1,\ldots,P_n\}$ と n 個の職 $\{W_1,\ldots,W_n\}$ がある．社員 P_i を職 W_j につけた場合，会社に c_{ij} の利益をもたらすことが分かっている．総利益を最大にするような社員と職の組合せ方を求める問題を割当問題という．この問題は，図 2.2 のようなヒッチコック型輸送問題で $n=m, a_i=1, b_j=1$ の場合にあたる．また，この問題は図 2.3 のような細工をすることにより，最小費用流問題の特殊な場合とみることもできる．供給点 $\{S_i\}$ の前に点 S を置き，リンク (S,S_i) を作り，その容量を a_i とする．また，輸送単価はゼロとする．需要点 $\{D_j\}$ の後に点 D を置き，リンク (D_j,D) を作りその容量を b_j とする．また，輸送単価はゼロとする．このグラフ上で S から D にいたる流量 $(\sum_{i=1}^{n} a_i)$ の最小費用流を作る問題となる．

最小費用流問題を解く主・双対法を割当問題に適用した場合，次の手順にまとめることができる．

1. 行列 C の初期値を利益行列 $[c_{ij}]$ とする．
2. $i=1$ から n まで次を行なう． C の第 i 行の成分の最大値を見つけて，その値を第 i 行の各成分から引く．
3. $j=1$ から n まで次を行なう． C の第 j 列の成分の最大値を見つけて，その値を第 j 列の各成分から引く．現在の C 行列を $\bar{C}=[\bar{c}_{ij}]$ とかく． $\bar{C}$ は各行，各列とも少なくとも 1 個のゼロ成分をもっている．
4. 最大流を求めるラベリング法を用いて， $\bar{C}$ のゼロ成分のリンクだけからなるグラフの最大流を求める．ただし前と後につける人為的なリンクの容量はすべて 1 とする．ラベリング法とは，グラフにおけるすべてのノードに（到着した親ノード，最短距離）のラベルを与えることで最適な経路を求める方法である．ここで，

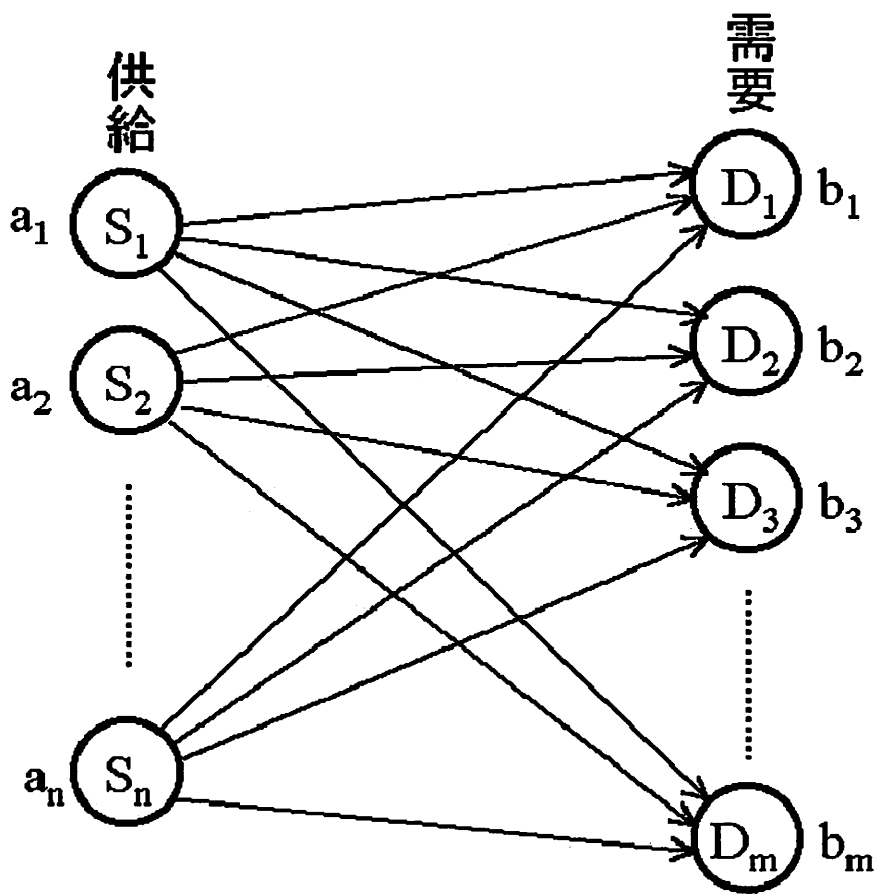

図 2.2: ヒッチコック型輸送問題

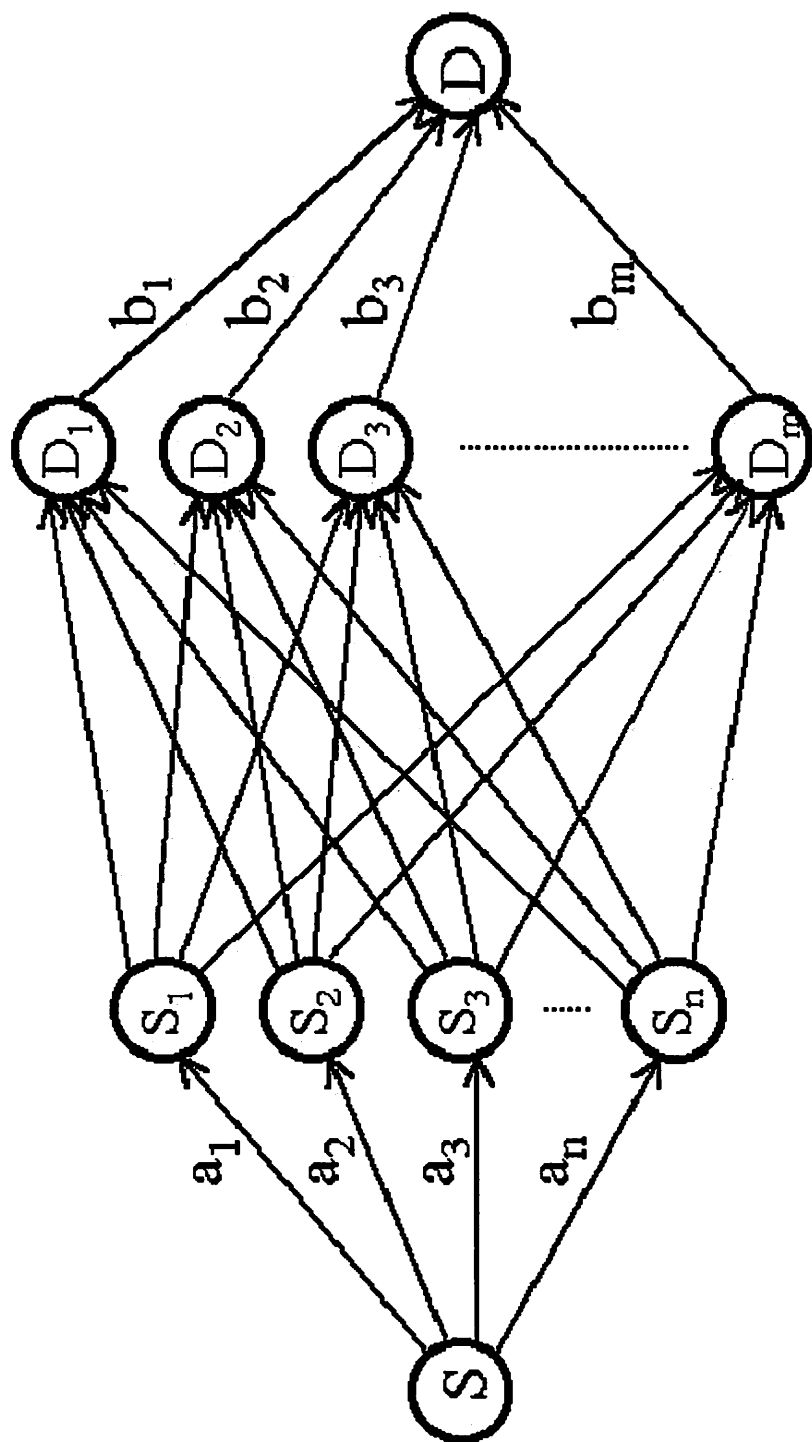

図 2.3: 輸送問題

- もし最大流が n ならば，現在の流れが最適割当を表わしている．
- もし最大流が n より小ならば手順5に進む．

5. 手順4の最大流が得られたときにラベルのついていない行（人）とラベルのついている列（職）を行列 $\bar{C}$ から隠し，残りの $\bar{C}$ 成分から引き，行と列の二重に隠された成分にはそれを加える．こうして新しい $\bar{C}$ 行列を作って，手順4に戻る．

2.2.2 巡回セールスマン問題

n 個の都市 $C_1, C_2, \ldots, C_n$ があって，C_i から C_j への距離が d_{ij} であるとき，すべての都市をちょうど1回ずつ巡回する最短閉路を見出す問題を巡回セールスマン問題という．

巡回路の総数は $(n-1)!$ だけあり，ちなみに $10! \simeq 3.6 \times 10^6, 20! \simeq 2.4 \times 10^{18}$ であることを見れば，片っ端から調べていくというやり方の破滅は目に見えている．この問題は実際面の応用範囲も広いので，いろいろな解法が提案されているが，それらを分類すれば (a) 発見的方法，(b) 動的計画法，(c) 分枝限定法になる．

ここでは (c) の方法を説明する．分枝限定法は最適化問題一般に適用できる原理であるが，とくに組合せ計画法の解法として適している．

分枝限定法では，組合せ問題のかわりに一連のより容易な問題を解く．巡回セールスマン問題の場合，かわりに巡回条件をはずした割当問題を解く．このとき，巡回セールスマン問題を原問題，割当問題を代替問題とよぶことにする．代替問題は原問題の条件を緩和したものにあたり，解法が容易でなければならない．その条件の緩和に応じて様々な代替問題がありうる．はじめに原問題のかわりに代替問題の最適解を求める．もしそれが原問題の条件をも満たしていれば，それで問題は解けたことになる．そうでない場合は，原問題の可能解をいくつかの集合に分割する．これが分枝とよばれる過程である．その分割した可能解の部分集合の

1つを取り出しその中で代替問題の最適解を求める．その解が原問題の可能解になっていれば，その部分集合の最適解でもあり，その部分集合は測深されたという．さもなければ，その部分集合をさらに分割していく．もし測深されたならば，その目的関数値は原問題の最適値の1つの上界値となるし，そうでない場合，代替問題の最適値はその部分集合に属する解全体の下界値となる．いま分枝限定法のある段階にいるものとする．可能解の集合 S が，$S_1, S_2, \ldots, S_k, S_{k+1}, \ldots, S_l$ に分割されており，そのうち $S_1, \ldots, S_k$ がすでに測深されているものとする．その測深された値の最小値 $\bar{z}$ は現在までに得られている最良（小）の目的関数値である．まだ測深されていない $S_{k+1}, \ldots, S_l$ のうち，この上限値 $\bar{z}$ より大きな下界値をもつものの中には，原問題の最適解はないことは明らかであるから以後無視してよい．これが，限定過程である．より小さな上界値 $\bar{z}$ が得られれば，より多くの集合が限定されて除去されるのである．未測深の集合から有望なものを1つ選んで分枝過程に入る．その結果を見て，限定過程や分枝過程を繰り返し，残りの部分集合がすべて限定され除去されるか，測深されてしまえば，その段階で原問題の最適解が得られたことになる．以上が分枝限定法であるが，実際問題として，次の戦略が検討されねばならない．

- 分枝をどの部分集合から始めるか ： それによって最適解を得るまでに分割しなければならない部分集合の個数が大いに異なる．
- 下界値をどのようにして計算するか ： なるべく大きな下界値が得られれば除去される部分集合が多くなって，探す手間は減少する．

2.2.3 貨物輸送問題（ナップサック問題）

遠足にもっていくナップサックの容積は $b\ cm^3$ である．この中に入れていく候補の品物には $G_1, \ldots, G_n$ の n 種類があり，それぞれ大きさは $a_1, \ldots, a_n\ cm^3$ である．また品物 G_j を持っていった場合，期待される価値は c_j である．価値の総和

を最大にするには，どの品物をナップサックにつめていけばよいか．これがナップサック問題である．

容積 b が十分大ならば全部の品物を持っていけるところだが，そうでない場合，どれとどれを持っていこうかと頭を悩ます種類の問題である．この問題を数式化するために，各 G_j に対して変数 x_j を設定し，G_j を持っていく場合 $x_j = 1$，持っていかない場合 $x_j = 0$ に対応させることにする．そのとき，ナップサック問題は次のように定式化できる．

$$\left.\begin{array}{ll} \text{最大化} & \displaystyle\sum_{j=1}^{n} c_j x_j \\ \text{制約条件} & \displaystyle\sum_{j=1}^{n} a_j x_j \leq b \\ & x_j \in \{0,1\},\ j = 1,2,\ldots,n \end{array}\right\}$$

この問題の特徴は変数 x_j が 0 または 1 という値しかとれないという点にある．変数（の一部）が整数値しかとれないという制約が加わった数理計画法を一般に整数計画法とよぶが，その中でも特に変数が 0 または 1 に限る場合を 0-1 計画法という．上のナップサック問題は制約条件が 1 個しかないという意味でその最も簡単な例である．

この問題でかりに 0-1 制約のかわりに $0 \leq x_j \leq 1$ としたならば，普通の線形計画法として簡単に解が得られる．かりに c_i/a_i が

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \ldots \geq \frac{c_n}{a_n}$$

の順になっているものとする．このとき，G_1 から順にナップサックにつめていき，スペースがあって丸ごと入れば 1 個入れる．もし最後に入れようとする品物がスペース不足で 1 個丸ごと入らなければ，その一部分（スペース分）だけを入れる．これが最適な詰め方であるが，最後の品物がちょうど 1 個入れば，それが 0-1 計画法の最適解であるとしても，そうでない場合には端数の解が得られることになって 0 か 1 かという問題の条件にあわなくなる．

ここでは，この問題を動的計画法を用いて解く．

$b, a_i (i = 1, \ldots, n)$ は整数値であると仮定する．次の関数を考える．$G_k(y)$ ： 容積 y の中に詰める品物の候補として $G_1, \ldots, G_k$ までに限定した場合に期待できる最大の価値．

こうすればナップサック問題の最適解は $G_n(b)$ を与える解ということになる．それを求めるために， $G_k(y)$ が満足するはずの次の関数に注目する．これを動的計画法の漸化式とよぶ．

$$G_k(y) = max\{G_{k-1}(y),\ G_{k-1}(y - a_k) + c_k\}$$

ここに $k = 1, 2, \ldots, n;\ y = 0, 1, \ldots, b$ であるが，$G_0(y) = 0 (y \geq 0)$，$G_k(y) = -\infty (y < 0)$ とする．この式の意味は次の通りである．容積 y の中に品物 $G_1, \ldots, G_k$ を候補として入れる場合に期待できる最大価値は，[y の中に $G_1, \ldots, G_{k-1}$ を候補として入れる場合の最大価値 $G_{k-1(y)}$] と [y の中にまず G_k を入れ，残りの $y - a_k$ の中に $G_k, \ldots, G_{k-1}$ を候補として入れる場合の最大価値 $G_{k-1}(y - a_k) + c_k$] の最大値に等しい．この式は $G_k(\bullet)$ の問題を $G_{k-1}(\bullet)$ の問題に帰着させる漸化式であり，解は $G_1(1), G_1(2), \ldots, G_1(b);\ G_2(1), \ldots, G_2(b); \ldots; G_n(b)$ の順に得られるのである．

第3章　ビタビアルゴリズム

3.1　自然言語処理とのかかわり

自然言語処理の個々の問題は，様々な手法を利用して，ある最適化問題に変換され，その問題を解くことで，当初の問題の解を得ることがしばしば行われる．例えば，形態素解析は形態素グラフの最小コストパスを求める問題に帰着する．日本語の係り受け解析も，文節同士の係る確率を求め，文全体の係り受け関係の確率を最大にする問題に帰着する．その他，固有表現抽出などで広く利用されるHMMでの状態遷移列の推定などもグラフの最小コストパスを求める問題に帰着する．これらの問題はすべて動的計画問題であり，動的計画法で解決できる．

ビタビアルゴリズムは動的計画法の一種であり，解析時間が短いという長所がある．そのため自然言語処理の分野では，ビタビアルゴリズムによって，それらの動的計画問題を解くことが一般的である．

3.2　問題の設定

図3.1のようなグラフが与えられているとする．グラフはいくつかのノードとそれを結ぶリンクから成り，各ノードとリンクにはコストが与えられている．ただし，サイクルは存在しない．SからEまでのパスのうちコストが最小となるようなパスを求めることを考える．これは，動的計画問題である．

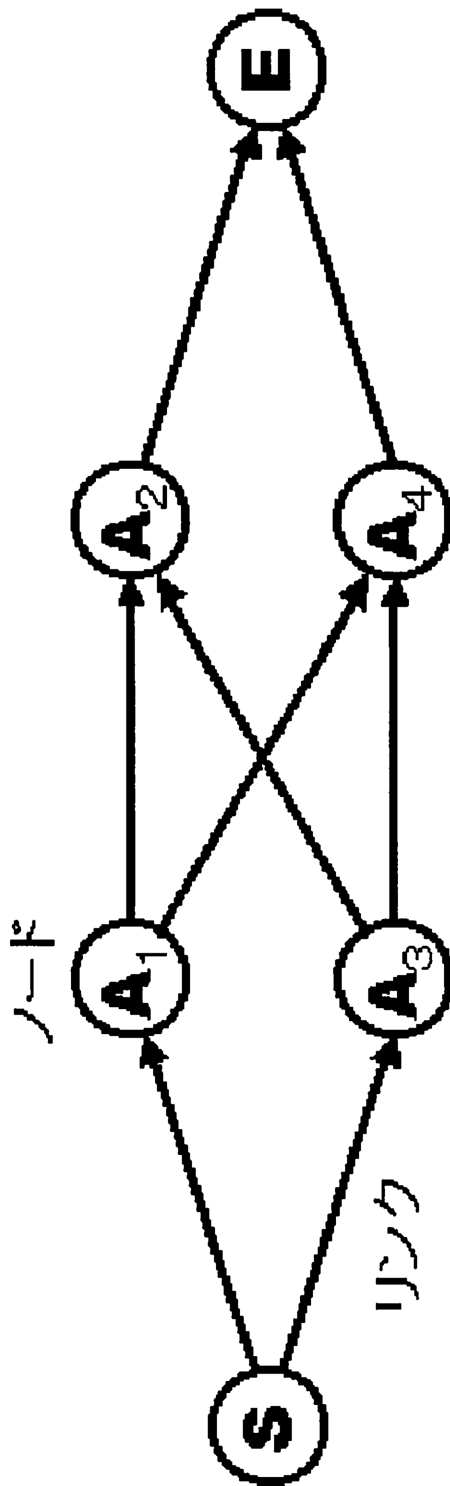

図 3.1: グラフの例

3.3 アルゴリズム

先に述べた動的計画問題は以下のビタビアルゴリズムで解くことができる [3].

1. ノードの位置を示すポインターを用意する．初期状態としてポインターを S の子ノード におく．
2. 親ノードを $p_1, p_2, \cdots, p_n$ 子ノードを $c_1, c_2, \cdots, c_m$ とする．各 c_i について左に連結可能な p_j を求め，c_i との間にリンクをはる．このとき
   - p_j までの部分最小コスト,
   - p_j, c_i 間の連結コスト,
   - c_i の単語コスト

   の和が最小であるような p_j を求め，その最小値を c_i までの部分最小コストとし，p_j と c_i とのリンクに特別のマークをつける．
3. ポインター位置から右側を順番に調べ, 次のノードまでポインターを移動する．
4. ポインターがノードの最後にくるまでステップ (2),(3),(4) を繰り返す．E の位置で終っているノードと E との連結可能性を調べ，可能なものだけをノードにリンクして処理を終了する．
5. 最終的にマークの付いたリンクによる S から E までのパス (ノードとリンクの並び)

   をコスト最小の解とする．

第4章　Javaによるビタビアルゴリズムの設計

4.1　Java言語

4.1.1　Javaについて

Java言語で作成したプログラムは，プラットフォームに依存せずに実行できるのが大きな特徴である．つまり，SolarisやWindows，Macintoshなどの広く普及しているOSはもちろん，ネットワーク端末などJavaをサポートしている機器であれば同じプログラムを動作させることができる．

Java言語のプログラム作成には，Java Developer's Kit(JDK)と呼ばれる開発環境を利用する．JDKはJavaの開発元であるSun Micro systems,inc.が配布しているもので，コンパイラ，インタープリタ，ライブラリなどが含まれる．JDKのコンパイラは，ソースプログラムをもとにバイトコードというプラットフォームに依存しない中間コードを生成する．このコードはJavaのインタープリタが解釈できるものである．したがって，Java言語で作成したプログラムは，Javaインタープリタをサポートしているマシン，すなわちJavaの環境があるすべての機器で，動作することになる．

また，Javaインタープリタの代わりにWWWブラウザソフトウェアでJavaのプログラムを実行することもできる．この場合はJavaをサポートしているブラウザ，すなわちJavaインタープリタを内蔵したブラウザを使用する．HotJavaやNetscapeは，こうしたブラウザの代表的なものである．

ブラウザを使用すると，Java言語で記述されたプログラムをネットワーク経由でロードし，ブラウザ内で実行することができる．

4.1.2 Java言語の特徴

Java言語はこれまでのコンピュータ言語には見られなかったほど大きな注目を集め，急速に受け入れられた．これは以下のような特徴を持っていたからである．従来の言語の利点を備え，欠点を解消したバランスの良さがJavaの最大の特徴ともいえる．

シンプル

- C++言語によく似た構文
- 自動ガーベッジ・コレクタ機能
- ポインタ機能を排除
- プログラムサイズがたいへん小さい

オブジェクト指向

- オブジェクトベースの効率的な開発環境

分散型

- ネットワーク経由でプログラムを実行可能
- TCP/IP用のライブラリを用意している

安定性

- コンパイル，実行時に厳しいエラーチェックが行われる
- 型のチェックやメモリの不正使用など，バグの原因を早期に検出可能

セキュリティ

- コンパイル，実行時に厳しいチェックがある

プラットフォーム非依存

- 作成したプログラムコードはアーキテクチャに依存しない
- 複数のプラットフォームをサポート
- データ型は標準化されている

インタープリタ

- インタープリタ形式の言語

マルチスレッド

- 言語レベルでマルチスレッド機能をサポート

4.1.3 オブジェクト指向

クラスとオブジェクトは，すべてのJavaプログラムの構築ブロックを形成している．したがって，これらについて基本的なことを理解しておく必要がある．

オブジェクトとは，状態と動作の両方を定義する記憶領域のことである．記憶領域はメモリであることもディスクであることもある．状態は，一連の変数とその中の値によって表現される．動作は，一連のメソッドとそれによって実装されるロジックによって表現される．したがって，オブジェクトとは，データとそれを操作するコードの組み合わせであると言える．

クラスとはオブジェクト作成の基盤となるテンプレートのことである．つまり，オブジェクトはクラスのインスタンスであると言える．新しいオブジェクトを作成する仕組みのことをインスタンス化と呼ぶ．

オブジェクト指向プログラミングの基盤となる3つの言語機能として，カプセル化，継承，ポリモーフィズムがある．ここでは，これらの3つの機能について簡単に説明する．

カプセル化

カプセル化とは，データとデータを操作するコードを関連付けるメカニズムである．コードは，データの周りに施される保護カプセルのようなものと考えることができる．ほかのソフトウェアからこのデータに直接アクセスすることはできない．

カプセル化には重要な利点がある．第一に，情報を保存するために使用するデータの形式を簡単に変更することができる．たとえば，リンクリストを使って情報を保存しており，パフォーマンスを向上させるために，後からこれをハッシュテーブルに変更することになったとする．このデータ構造へのすべてのアクセスが，厳密に定義された一連のアクセスメソッドによって行われていれば，これらのメソッドを呼び出すほかのソフトウェアを修正することなく，データ構造に修正を加えることができる．第二に，機密情報へのアクセスを調整するのがとても楽になる．たとえば，データの特定の部分を修正するメソッドでは，ログインとパスワードを受け取るようにすることができる．第三に，複数のスレッドからのデータアクセスを同期化することがとても簡単になる．

継承

継承とは，あるクラスを特化することによって，ほかのクラスを定義するメカニズムのことである．1つのクラスによってすでに定義されている構造体や動作を基に別のクラスを定義することができるため，ソフトウェアの再利用性が促進される．クラスYがクラスXを継承する場合，Yを「Xのサブクラス」と呼び，X

を「Yのスーパークラス」と呼ぶ．また,「YはXを拡張する」と表現することもある．さらに，ほかのクラスがYを拡張することもできる．このようにして，一連のクラスにに対して継承の階層関係が形成される．

1つのクラスは複数のサブクラスを持つことができる．しかしJavaでは，1つのクラスは直接的なスーパークラスを1つしか持つことができない．継承階層のルートにはObjectという名前のクラスがある．したがって，すべてのJavaクラスはObjectクラスを直接的または間接的に継承していることになる．

これらの点を理解するために，図4.1のクラス階層を考えてみる．階層のルートにはObjectクラスがある．Objectクラスのサブクラスとして A，B，C といったクラスがある．クラスAにはD，E，Fというサブクラスがある．クラスCはクラスGのスーパークラスである．クラスAがa1，a2，a3というメソッドを持っているとすると，これらのメソッドはクラスAのすべてのサブクラスに継承される．

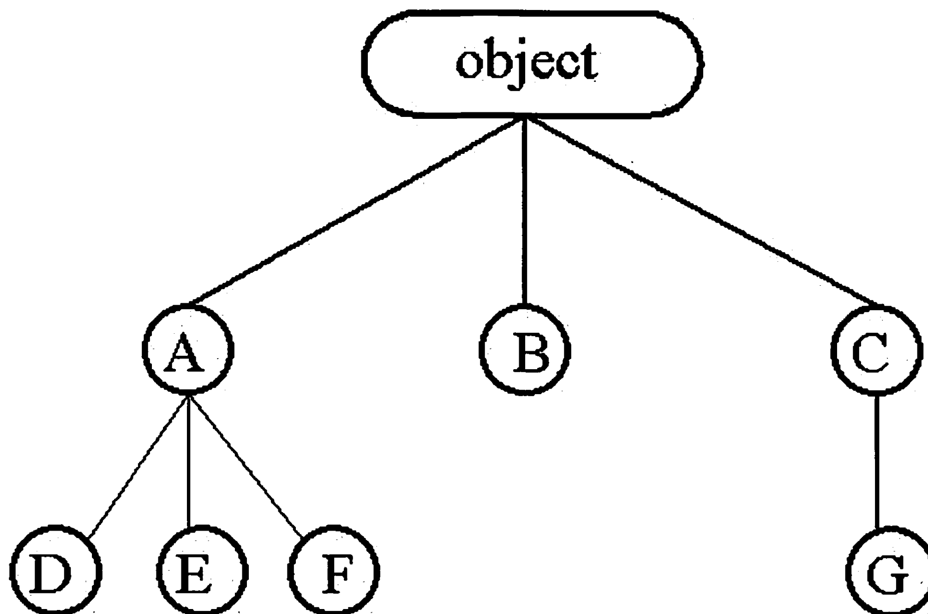

図 4.1: クラス継承階層

ポリモーフィズム

ポリモーフィズムとは,「1つのインターフェイス，複数の実装」と表現されるメカニズムである．図4.2に示すクラス階層について考えてみる．この階層では,クラスShapeがEllipse，Rectangle，Triangleという名前のサブクラスを持っている．これらはそれぞれ各種の図形(Shape)を表している．

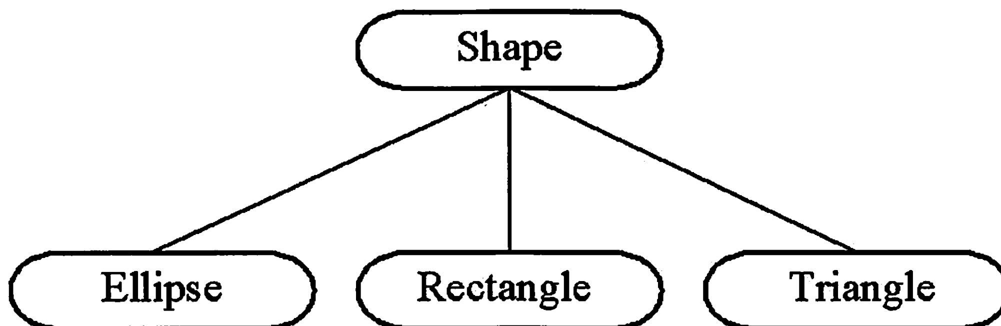

図4.2: ポリモーフィズム

ShapeクラスではgetArea()というメソッドが定義されている．これは図形の面積を返すメソッドである．しかし，このメソッドはこのクラスによって実装されていない．したがって，これは抽象メソッドであり，Shapeは抽象クラスである．つまり，Shapeクラスのオブジェクトを作成することはできない．しかし，このクラスの機能はサブクラスによって継承することができる．

Shapeクラスの各サブクラスでは，getArea()メソッドを実装する．ただし，面積の計算方法はEllipse，Rectangle，Triangleの各オブジェクトによってまったく異なる．したがって，Shapeクラスのオブジェクトがいくつかあり，それらの総面積を計算する必要がある場合は，これらの各オブジェクトのgetArea()メソッドを呼び出すことになる．Javaは，各オブジェクトの適切な実装を確実に呼び出す．これは実行時に行われ，ポリフォーリズムの実例を示している．

ここで，オブジェクト指向プログラミングの大きな利点の1つを紹介する．各

種のShapeクラスを管理するための100万行ものJavaコードがすでに存在すると仮定する．そして，Trapezoidという名前のShapeクラスの新しいサブクラスを作成する．Trapezoidサブクラスでも，getArea()メソッドを実装する．このとき，既存の100万行のコード内を検索し，このソフトウェアを更新する必要はない．Trapezoidクラスはすべての Shapeクラスが持っているものと想定されるメソッドを実装しているので，それを操作するソフトウェアを修正しなくても正しく動作する．

4.2 Javaを用いる利点

ビタビアルゴリズムなどの動的計画法は，電子計算機の利用と結びつき，適用される部門が広範囲，かつ多方面にわたっている．そのため，利用されているプラットフォームもさまざまである．プラットフォームに依存しないプログラムを作成することは汎用性を高めることにつながる．Javaにより作られたアプリケーションは，プラットフォームに依存しないため，汎用性の高いビタビアルゴリズムを作ることができる．

また，Javaはオブジェクト指向言語であるため，問題をグラフ構造に置き換えて解くビタビアルゴリズムの作成に向いている．さらに，継承の性質により，クラスを拡張させ，さまざまな問題に適応させることが容易となる．

おまけに，GUIのライブラリが最初から含まれているのでグラフの表示が簡単に行える．それにより，視覚的に問題の解を得ることができる．

第5章　種々の動的計画問題への適用

先に述べたビタビアルゴリズムを用いて，まず形態素グラフの最適パスを求めるプログラムを作成する．次に，そのプログラムを拡張して代表的な動的計画問題を解く．

今回は，C言語で作成したビタビアルゴリズムのプログラムをJava[4]で書き換え，それぞれの問題に対応させたプログラムを作成する．

5.1　形態素グラフの最適パスを求める問題

5.1.1　問題例

「このひとことで元気になった」という文は，辞書引きと品詞間の接続表を用いることで，図5.1のような形態素グラフに変換できる．ノードやリンクのコストは，辞書や接続表から得られる．この文の形態素解析はこの形態素グラフから最小コストパスを求めることで行われる．

図5.1では，( )内の数字が各ノード，リンクまでの最小コストで，太線のリンクがマーク付けされたリンク(部分最適解を示すリンク)を示している．

文の始めと終りには，それぞれ < 文頭 > (Sノード)，< 文末 > (Eノード)という仮想的なノードがおかれている．これは図3.1で示したグラフの類であり，ビタビアルゴリズムによってコスト最小のパスを得ることができる．

図 5.1: ビタビアルゴリズムによるコスト計算

5.1.2 プログラムの構成

先に述べたアルゴリズムを実装するためにまず次の2つの要素を設定する.

1つめは，GraphNodeクラスである．グラフのノードを1つのクラスで定義して，実際のグラフのノードをこのクラスのインスタンスとして実現する.

2つめは，viterbiEngineメソッドである．このメソッドは，グラフを逆から辿り，最適な親ノードを見つけてそこまでの合計コストを計算する部分である.

また今回は，実行時に動的に大きくなったり小さくなったりするようなデータを格納する必要があるため，配列ではなく，リストリンクを使う．これを使うことで，データをより速く検索することが可能であり，プログラミングも単純化され，必要なメモリ容量も抑えることができる.

リストリンクは一連の要素（ノード）から構成されている．また，各要素はデータおよび別のノードに接続するためのリンクから構成されている．図5.2に単純なリストリンクの例を示す.

リストリンクをJavaで作成するには，データと次の要素へのリンクを含むようなクラスを設定する．今回作成したプログラムでは，ノードのデータとグラフのリンクコストを格納するListNodeクラスを設定した.

5.1.3 データファイルの表現

グラフに対応するデータファイルを作成する．そのデータを表5.1に示す．ここで，*id*はノードにつけた番号，*pos*は入力文中における単語の位置を表す.

このデータから，作成したビタビアルゴリズムのプログラムを使い以下のような単語分割の結果が得られた．また，最小コストは230となった．これはあらかじめ計算した値と一致した.

|この|ひとこと|で|元気|に|なった|

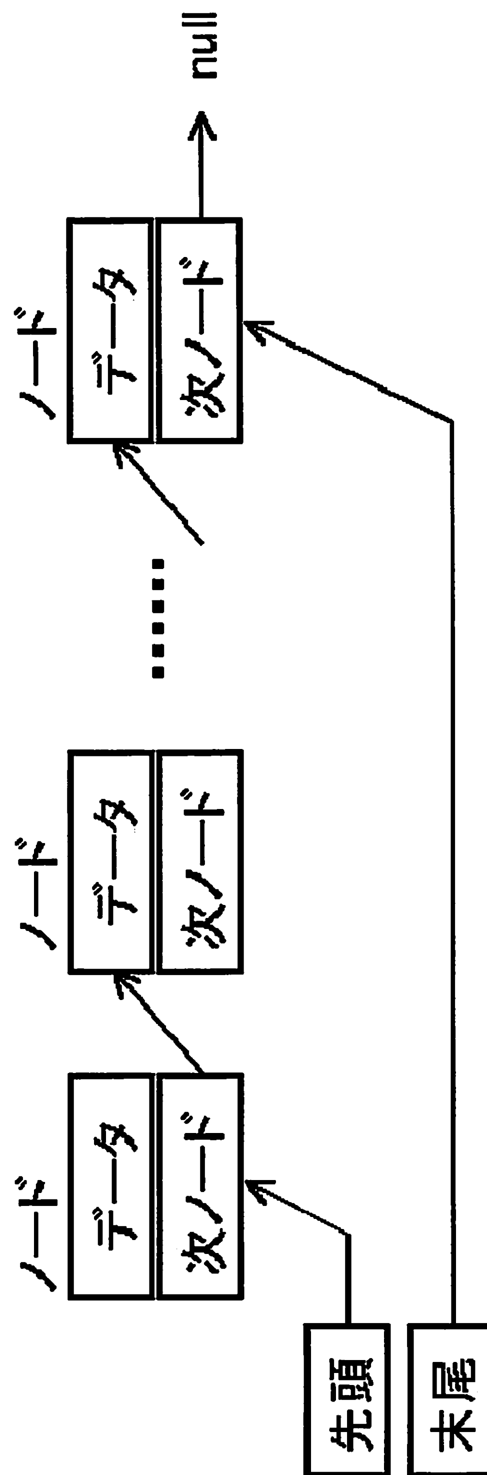

図 5.2: 単純なリストリンクの例

表 5.1: データの形式

id	文字列	*pos*	単語コスト	子ノード (連接コスト)
0	< 文頭 >	0	0	1(10)
1	この	1	10	2(10),3(10),4(10)
⋮	⋮	⋮	⋮	⋮
11	に	10	10	12(10)
12	なった	11	40	13(10)
13	< 文末 >			

5.2 その他の動的計画法への拡張

5.2.1 割当問題

グラフの表現

利益行列 $\mathbf{C} = (c_{ij})$ をもつ割当問題を次に示す.

$$\mathbf{C} = \begin{pmatrix} 0 & 6 & 10 & 2 & 3 \\ 9 & 0 & 10 & 5 & 7 \\ 8 & 6 & 0 & 4 & 1 \\ 3 & 4 & 9 & 0 & 7 \\ 6 & 5 & 2 & 7 & 0 \end{pmatrix}$$

ここで，c_{ij} は社員 P_i を職 W_j につけた場合の利益とする．これをビタビアルゴリズムで解くために，図 5.3 に示すようなグラフを考える．この問題では，リンクにおけるコストは必要ないので，リンクコストをすべて 0 とする．またこの問題は，形態素グラフの最適パスを求める問題と異なり，同じ名前のノードが多数存在するため一度通ったノードと同じ名前をもつノードは通らないようにする．

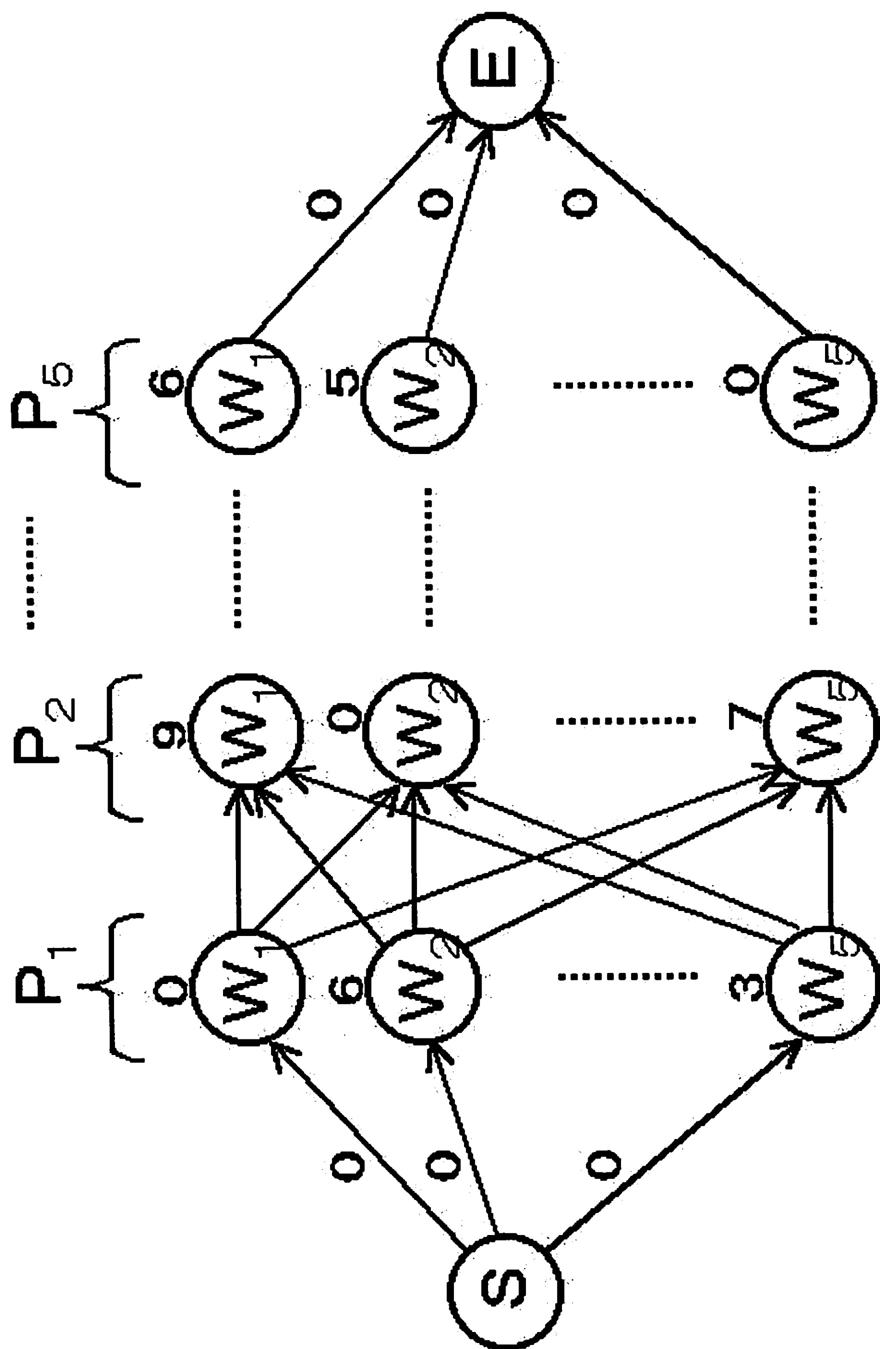

図 5.3: 割当問題のグラフ表現

表 5.2: 割当問題におけるデータの形式

id	職種	*pos*	利益	子ノード (連接コスト)
0	< S >	0	0	1(0),2(0),...,5(0)
1	W1	1	0	6(0),7(0),8(0),...,10(0)
2	W2	2	6	6(0),7(0),8(0),...,10(0)
⋮	⋮	⋮	⋮	⋮
25	W5	5	0	26(0)
26	< E >	0		

データファイルの表現

図5.3のグラフを踏まえて表5.2に示すようなデータファイルを作成する．各ノードのコストなどは形態素グラフの最適パスを求める問題で作成したデータファイルと同様に入力する． また，ここでの *id* はノードの通し番号， *pos* は職種を表す番号とする．

プログラムの拡張

この問題は，形態素グラフの最適パスを求める問題と異なり同じノードが多数存在するため，一度通ったノードと同じ名前のノードを通らないようにプログラムを拡張する必要がある．

5.2.2 巡回セールスマン問題

グラフの表現

コスト行列 $\mathbf{D} = (d_{ij})$ をもつ割当問題を次に示す．

$$\mathbf{D} = \begin{pmatrix} \infty & 97 & 60 & 73 & 17 & 52 \\ 97 & \infty & 41 & 52 & 90 & 30 \\ 60 & 41 & \infty & 21 & 35 & 41 \\ 73 & 52 & 21 & \infty & 95 & 46 \\ 17 & 90 & 35 & 95 & \infty & 81 \\ 52 & 30 & 41 & 46 & 81 & \infty \end{pmatrix}$$

ここで，d_{ij} は都市 C_i から都市 C_j までの距離とする．

これをビタビアルゴリズムで解くために，図 5.4 に示すようなグラフを考える．この問題では，各ノードにおけるコストは必要ないので，ノードのコストをすべて 0 とする．また，＜S＞から C_j，C_i から＜E＞のリンクコストも同様に 0 とする．また，割当問題と同様に，一度通ったノードと同じ名前をもつノードは通らないようにする．さらに，この問題は始点と終点にあたるノードを一致させる必要があるので，終点にあたるノードにだけ始点にあたるノードと終点にあたるノード間のリンクコスト（距離）を与える．

データファイルの表現

図 5.4 のグラフを踏まえて表 5.3 に示すようなデータファイルを作成する．この問題も割当問題の場合と同様に id はノードの通し番号，pos は都市の種類を表す番号とする．行列内にある ∞ はデータファイル内では，十分に大きな値として 1000 を与える．

プログラムの拡張

この問題も割当問題と同様に，一度通ったノードを通らないようにする．また，最初と最後のノードを一致させる必要もある．

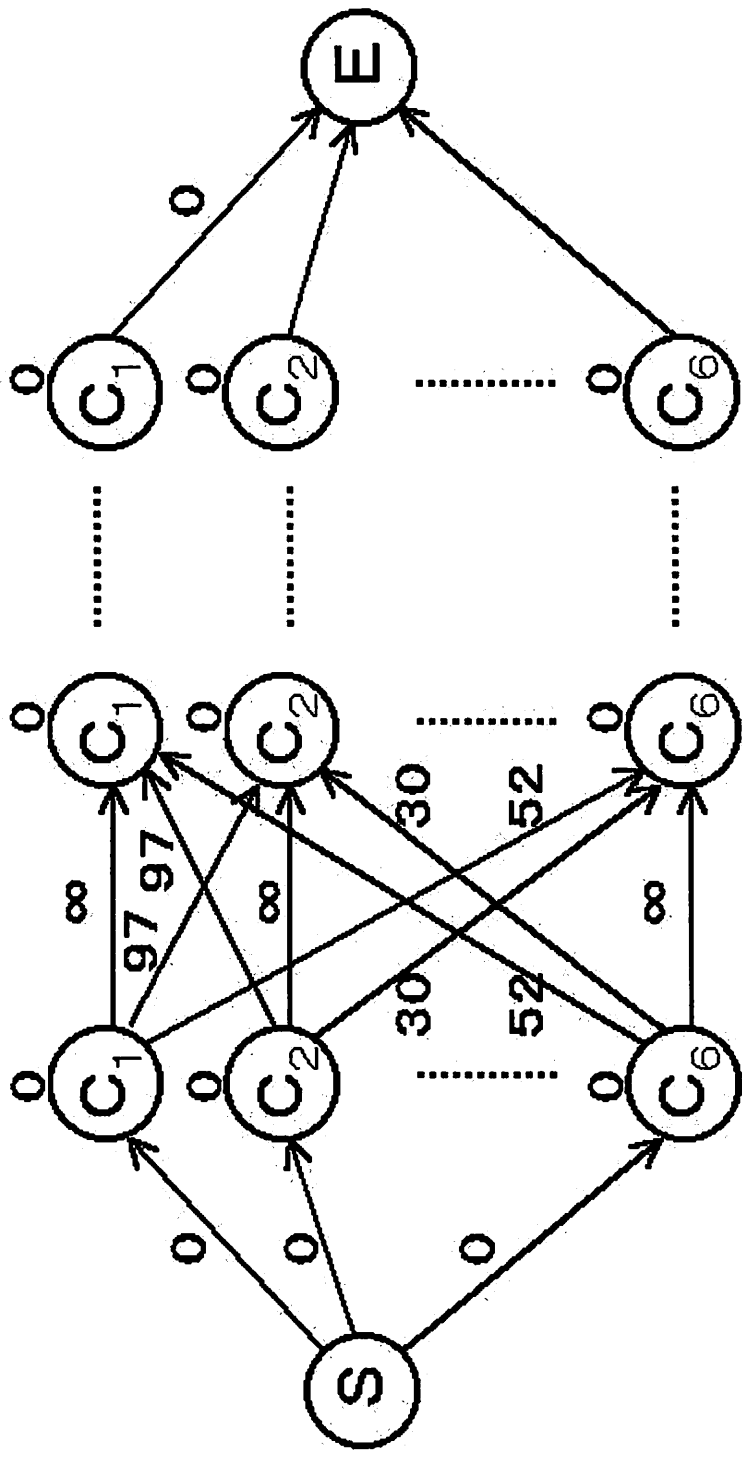

図 5.4: 巡回セールスマン問題のグラフ表現

表 5.3: 巡回セールスマン問題におけるデータの形式

id	都市	*pos*	ノードコスト	子ノード (連接コスト)
0	< S >	0	0	1(0),2(0),3(0),. . . ,6(0)
1	C_1	1	0	7(1000),8(97),9(60),. . . ,12(52)
⋮	⋮	⋮	⋮	⋮
42	C_6	6	0	43(0)
43	< E >	0		

5.2.3 貨物輸送問題（ナップサック問題）

グラフの表現

問題例を次に示す.

ここでの，ナップサックの容積は 25 であるとする.

品物 i	1	2	3	4	5	6	7	8
容積 a_i	2	5	18	3	4	5	10	2
価値 c_i	5	10	13	4	10	11	13	3

これをビタビアルゴリズムで解くために，図 5.5 に示すようなグラフを考える. このグラフは，今までのものとは異なり，各ノードに品物の容積にあたる a_i と，価値にあたる c_i の 2 種類のコストを定義する．図には, (a_i ， c_j) として示している.

また，問題で与えられたノードとは別に， D という新しいノードを設ける．このノードは仮想的なノードであり，容積 a_i と価値 c_i を共に 0 とする．品物の容積 a_i の合計がナップサックの容積 b を越えると選ばれるようにし，ノード D だけは何度でも通ることができるように設定する．その他のノードに関しては，前の 2 つの問題と同様とする.

表 5.4: ナップサック問題におけるデータの形式

id	品物	pos	容積 a_i	価値 c_i	子ノード (連接コスト)
0	< S >	0	0	0	1(0),2(0),3(0),...,9(0)
1	G_1	1	2	5	10(0),11(0),12(0),...,18(0)
⋮	⋮	⋮	⋮	⋮	⋮
72	D	9	0	0	73(0)
73	< E >	0	0		

この問題でも，リンクにおけるコストは必要ないので，リンクコストをすべて0とする．

データファイルの表現

図 5.5 のグラフを踏まえて表 5.4 に示すようなデータファイルを作成する．この問題も割当問題の場合と同様に id はノードの通し番号，pos は品物の種類を表す番号とする．

プログラムの拡張

この問題は，今までの問題と異なり，コストを重量とした場合，価格にあたるもう 1 つの新しい変数が必要となる．さらに，合計コストには上限を設けなければならない．

5.3 グラフノードのデータ構造

グラフの各ノードを GraphNode クラスで定義して，実際のグラフのノードをこのクラスのインスタンスとして実現する．GraphNode クラスには，図 5.6 に示

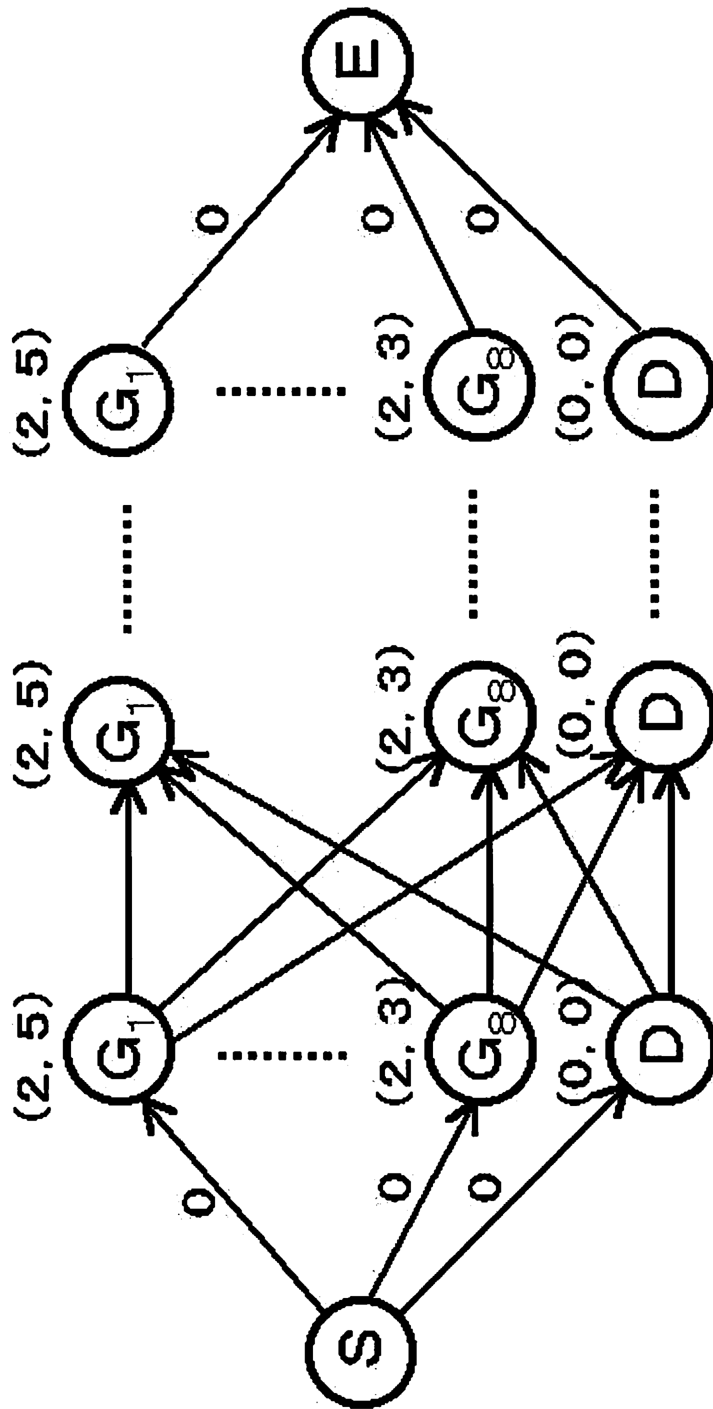

図 5.5: ナップサック問題のグラフ表現

すような変数が定義されている．この中には，ListNode でインスタンス化された parents と children が含まれており，これらはそれぞれ他の parents と children にリンクしている．

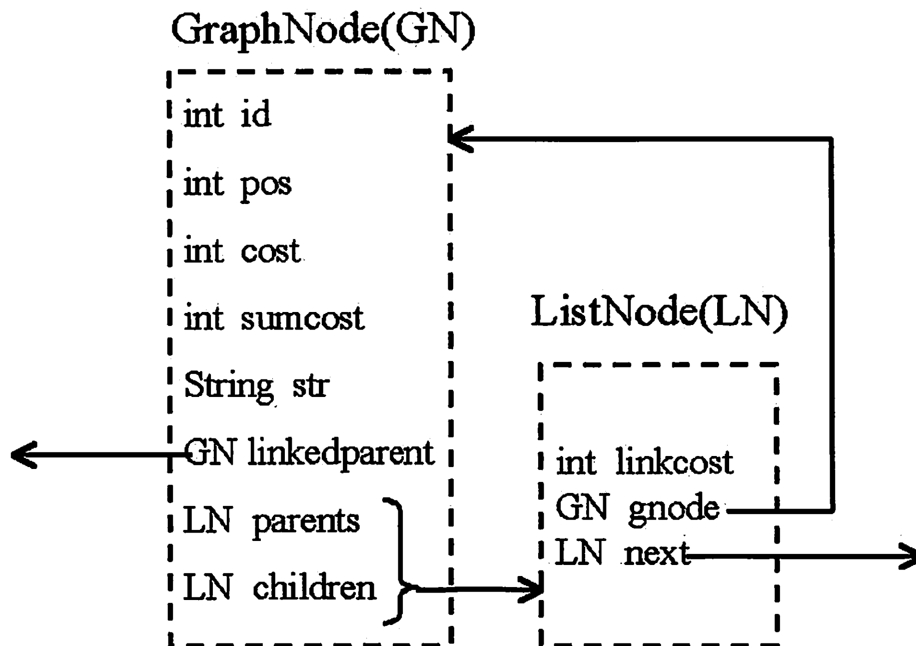

図 5.6: データ構造

第6章　考察

ビタビアルゴリズムで様々な動的計画問題が解けることは確認できたが，正しい問題を導くのが困難な問題も存在した．ここでは，当研究室の卒研 [5] の実験の一部に協力する形で，ビタビアルゴリズムで単語の並べ替え問題を 20 問解いたが，正解を出すことができなかった．例えば,「阪神大震災の教訓や難民支援の経験が生かされているのだろう.」という文を単語分割し，それをランダムに並べたものから元の文を復元することを試みたが，結果は「生かさ | れ | て | いる | の | が | 阪神 | 大 | の | 難民 | 支援 | の | 経験 | や | 震災 | 教訓 | だろ | う |.」となり, 正しい結果には至らなかった．これは，ビタビアルゴリズムが最適解を求めるのに大域的な関連性が効いてくるような動的計画問題には弱いためと思われる．

また，今回作成したプログラムにおいて viterbiEngine メソッドをクラスとして独立させるべきであった．それにより，問題によってそのクラスを継承した拡張クラスを設けることによって，さらに容易に問題が解けるようになると思われる．

第7章　おわりに

ビタビアルゴリズムの汎用的なツールを作成するために，まず形態素グラフの最小コストパスを求めるプログラムを実装させた．次に，そのプログラムを拡張して代表的な動的計画問題を解いた．これらのプログラムを作成するにあたり，Java言語を用いることでオブジェクト指向的なプログラムを実装できた．そのために，各問題への対応が容易になっている．

今後の課題としては，プログラムのエンジン部分をクラスとして独立させることで，より汎用性を高めることが挙げられる．また，自然言語処理に特化させ，効率化した自然言語処理用ビタビアルゴリズムツールの作成を目標としたい．

謝辞

本研究の遂行及び論文の作成において多大な御助言及び御指導を賜りました新納 浩幸 教官（茨城大学工学部システム工学科）に深い感謝の意を表します.

また，御指導を頂きましたシステム工学科計算機応用講座の教官の方々にも深く感謝致します.

最後に，本研究を進めるにあたり助言，協力を頂きました，同研究室の池谷 昌紀 氏（茨城大学大学院理工学研究科システム工学専攻），星 秀明 氏（茨城大学工学部システム工学科4回生），徳永 崇 氏（茨城大学システム工学科4回生），堤研究室の荒井 亮一 氏（茨城大学大学院理工学研究科システム工学専攻）に深く感謝致します.

関連図書

[1] 刀根薫：数理計画, 朝倉書店, (1978).

[2] 鍋島一郎：動的計画法, 森北出版株式会社, (1968).

[3] 長尾真 ：自然言語処理，岩波書店, (1996).

[4] Sun Microsystems,Inc. : JAVA プログラミング講座 ， 株式会社アスキー, (1996).

[5] 徳永崇 ：学習データ中のノイズによる言語モデルへの影響，H12 年度 茨城大学工学部システム工学科 卒業論文, (2001).

付 録A　プログラムリスト（C言語）

```
#ifndef __ATSU_H__
#define __ATSU_H__

//Header Files
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>

//変数の宣言
static char word[256][256],cnlist[256][256];
static int n,nodecost[256],startpos[256],
e[256][256],f[256][256],len;

//関数のプロトタイプ宣言
void read_data(FILE *fp);
void get_id_pos(int pos,int a[]);
void get_parent_ids(int l,int b[]);
int link_cost(int x,int y);

#endif

/****************************************************/

#include"atsu.h"

int main(int argc,char *argv[])
{
  FILE *fp;
  int k=0,q,pos,a[256],b[256],
```

```
    min_cost,cost,cost_ex=0,cost_lin,cost_no,sp=0,
stack[100],id_list[256],cost_list[256];
  extern  char word[256][256];
  extern int n,nodecost[256];
  int gomi;

  if(argc!=2){
    printf("argc=%d \n",argc);
    printf("引数の数が違います。 \n");
    exit(1);
  }

  if((fp=fopen(argv[1],"r"))==NULL){
    printf("cannot open!!\n");
    exit(1);
  }

  read_data(fp);

  for(pos=1;pos<=len;pos++){
    min_cost=1000;

    get_id_pos(pos,a);

    k=0;

    while(a[k]!=-1){

      get_parent_ids(a[k],b);
      q=0;
      while(b[q]!=-1){

cost_ex=cost_list[b[q]];
cost_lin=link_cost(b[q],a[k]);
cost_no=nodecost[a[k]];

cost=cost_ex+cost_lin+cost_no;

if(cost<=min_cost){
```

```
  min_cost=cost;
  cost_list[a[k]]=min_cost;
  id_list[a[k]]=b[q];
}
q++;
      }
      k++;
      min_cost=1000;
    }
  }

  sp = n;
  gomi = 0;

  while(sp!=0){
    stack[gomi] = sp;
    sp = id_list[sp];
    gomi++;
  }

  stack[gomi] = 0;
  stack[gomi+1] = -1;
  for(;gomi>=0;gomi--)
    printf("%d:%s",stack[gomi],word[stack[gomi]]);

  return(0);
}

void read_data(FILE *fp)
{
  int i=0;
  int l;
  int id=0;

  char pos_c[256],cost_c[256];
  int ggg,kkk;
  char abc[256];
  char m[256];
```

```
while(fgets(m,256,fp)!=NULL){
  for(i=0;m[i]!=' ';i++);

  for(;m[i]==' ';i++);
  l=i;
  for(;m[i]!=' ';i++)  word[id][i-l]=m[i];
  word[id][i-l]='\n';

  if(strstr(m,"文末")!=NULL) break;

  for(;m[i]==' ';i++) ;
  l=i;
  for(;m[i]!=' ';i++)  pos_c[i-l]=m[i];
  pos_c[i-l]='\0';
  startpos[id]=atoi(pos_c);

  for(;m[i]==' ';i++);
  l=i;
  for(;m[i]!=' ';i++) cost_c[i-l]=m[i];
  cost_c[i-l]='\0';
  nodecost[id]=atoi(cost_c);

  for(;m[i]==' ';i++);
  l=i;
  for(;m[i]!='\n';i++)  cnlist[id][i-l]=m[i];
  cnlist[id][i-l]='\n';

  id++;
}
n=id;
nodecost[n] = 0;

len = 0;
ggg = 1;

while(ggg<n){
  len += (strlen(word[ggg])-1)/2;
  strcpy(abc,cnlist[ggg]);
  for(kkk=0;abc[kkk]!='(';kkk++)
```

```
      if(kkk>strlen(abc)) break;
    abc[kkk] = '\0';
    ggg = atoi(abc);
  }
  fclose(fp);
}

void get_id_pos(int pos,int a[])
{
  int i=0,j=0;
  extern int n, startpos[256];

  if(pos==len){
    a[0] = len;
    a[1] = -1;
    return ;
  }
  else if(pos>len){
    printf("pos は%d 未満!!\n",len+1);
    exit(1);
  }

  for(i=0;i!=n;i++){
    if(pos==startpos[i]){
      a[j]=i;
      j++;
    }
  }
  a[j]=-1;
}

void get_parent_ids(int id,int b[])
{
  int i=0,j=0,k=0,l=0;
  char d[256];
  extern char cnlist[256][256];

  if(id>n){
    printf("id は%d 未満!!\n",n+1);
```

```
    exit(1);
  }

  for(i=0;i<n+1;i++){
    for(j=0;j<strlen(cnlist[i]);j++){
      for(l=0;cnlist[i][j]!=')';l++)
d[l]=cnlist[i][j];
      d[l]='\0';
      e[i][k]=atoi(d);
      k++;  l=0;
      for(;cnlist[i][j]!=')';j++);
      j+=2;
    }
    e[i][k]=-1;
    j=0; k=0;
  }  k=0;

  for(i=0;i!=n+1;i++){
    for(j=0;e[i][j]!=-1;j++){
      if(e[i][j]==id){
b[k]=i; k++;
      }
    }
  }
  b[k]=-1;
}


int link_cost(int x,int y)
{

  int i=0,j=0,k=0,l=0,z=0;
  char d[256];
  extern char cnlist[256][256];
  extern int e[256][256];

  for(i=0;i<n+1;i++){
    for(j=0;j+1<strlen(cnlist[i]);j++){
      for(;cnlist[i][j]!='(';j++);
```

```
      j++; //space ++

      l=j
      for(;cnlist[i][j]!=')';j++)
d[j-l]=cnlist[i][j];

      d[j-l]='\0';
      f[i][k]=atoi(d);
      k++;
    }
    f[i][k]=-1;
    k=0;
  }

  for(j=0;e[x][j]!=-1;j++){
    if(e[x][j]==y) z=f[x][j];
  }
  return(z);
}
```

付 録B　プログラムリスト（Java）

```
import java.io.*;

class Viterbi2 {

  public static void main(String args[]) {
    GraphNode gns[];
    BufferedReader br = null;
    int size=0;
    int num =0;

    try{
      br = new BufferedReader(new InputStreamReader
(new FileInputStream(args[0])));
    }
    catch(Exception e) { System.err.println(e);System.exit(1);}
    try{
      String s;
      while ((s=br.readLine())!=null){
size++;
      }
      br.close();
    }
    catch(Exception e) { System.err.println(e);System.exit(1);}

    gns = new GraphNode[size];

    for(int i = 0;i<size;i++) {
      gns[i] = new GraphNode();
    }
    gns[0].sumcost = 0;
```

```
    try{
      br = new BufferedReader(new InputStreamReader
(new FileInputStream(args[0])));
    }
    catch(Exception e) { System.err.println(e);System.exit(1);}

    String s;
    try{
      while ((s=br.readLine())!=null){
atsushi(gns,s,num);
num++;
      }
      br.close();
    }
    catch(Exception e){System.err.println(e);System.exit(1);}

    //for(int i=0;i<14;i++){
    //  gns[i].PrintForCheck();
    //}

    viterbiEngine(gns[size-1]);
    printMinPath(gns,size);

  }

  public static void viterbiEngine(GraphNode gn) {
    GraphNode minpn = getMinParent(gn);
    gn.linkedparent = minpn;
    gn.sumcost = minpn.sumcost + gn.cost + getLinkCost(gn,minpn);
  }

  public static GraphNode getMinParent(GraphNode g){
    ListNode gomi = g.parents;
    GraphNode gg = null;
    int min = 10000;
    System.out.println("aaa"+g.id);
    System.out.println("bbb"+gomi.gnode.id);

    while(gomi!= null) {
```

```
      while(gomi.gnode.sumcost == -1) {
viterbiEngine(gomi.gnode);
      }
      if(min > gomi.gnode.sumcost){
min = gomi.gnode.sumcost;
gg = gomi.gnode;
      }
      gomi = gomi.next;
    }
    return(gg);
  }

  public static int getLinkCost(GraphNode g,GraphNode minpn) {
    int lc = 0;
    ListNode gomi = g.parents;
    while(gomi != null) {
      if(gomi.gnode.id == minpn.id) {
lc = gomi.linkcost;
      }
      gomi = gomi.next;
    }

    return(lc);
  }


  public static void printMinPath(GraphNode gn[],int size) {
    int ids[] = new int[size];
    int i = 0;
    GraphNode gomi = gn[size-1];
    while(gomi != null){
      ids[i] = (gomi.id);
      gomi=gomi.linkedparent;
      i++;
    }
    for(int j= i-1;j>=0;j--)
      System.out.println(ids[j]+"  "+gn[ids[j]].str);
    System.out.println(gn[size-1].sumcost);
  }
```

```
  public static void atsushi(GraphNode gn[],String s,int num){
    StringBuffer strb1 = new StringBuffer(s);

    StringBuffer strb2  = new StringBuffer(strb1.toString());
    atsushi2(' ',strb1,strb2);

    StringBuffer strb3  = new StringBuffer(strb1.toString());
    atsushi2(' ',strb1,strb3);

    StringBuffer strb4  = new StringBuffer(strb1.toString());
    atsushi2(' ',strb1,strb4);

    if(strb1.length() == 0){
      gn[num].setnode(num,strb3.toString(),
Integer.parseInt(strb4.toString()),0);
      return;
    }


    StringBuffer strb5  = new StringBuffer(strb1.toString());
    atsushi2(' ',strb1,strb5);

    int k = atsushi3(',',strb1.toString());

    StringBuffer strb6[]  = new StringBuffer[k];

    if(k!=1){
      for(int i =0;i<k-1;i++){
strb6[i] = new StringBuffer(strb1.toString());
atsushi2(',',strb1,strb6[i]);
      }
      strb6[k-1] = new StringBuffer(strb1.toString());
    }
    else strb6[0] = new StringBuffer(strb1.toString());

    StringBuffer strb7[] = new StringBuffer[k];
    StringBuffer strb8[] = new StringBuffer[k];
```

```
    for(int i=0;i<=k-1;i++){
      strb7[i] = new StringBuffer(strb6[i].toString());
      strb8[i] = new StringBuffer(strb6[i].toString());
      atsushi2('(',strb8[i],strb7[i]);
      int j = strb8[i].length();
      strb8[i].delete(j-1,j);
    }

    gn[num].setnode(num,strb3.toString(),
Integer.parseInt(strb4.toString()),
Integer.parseInt(strb5.toString()));
    for(int i=0;i<k;i++)
      gn[num].setoyako(gn[Integer.parseInt(strb7[i].toString())],
Integer.parseInt(strb8[i].toString()));

  }

  public static void atsushi2(char c,StringBuffer st1,
                                            StringBuffer st2){
    String str = st1.toString();
    int i = str.indexOf(c);

    st2.delete(i,str.length());
    st1.delete(0,i+1);
  }

  public static int atsushi3(char c,String s) {
    int i = 0,b = 0;
    while(i!=-1){
      i = s.indexOf(c,i+1);
      b++;
    }
    return b;
  }

}

class GraphNode {
```

```
  int id;
  int pos;
  int cost;
  int sumcost;
  String str;
  ListNode parents;
  ListNode children;
  GraphNode linkedparent;

  GraphNode() {
    sumcost = -1;
  }

  void setnode(int id, String str, int pos, int cost) {
    this.id = id;
    this.str = str;
    this.pos = pos;
    this.cost = cost;
  }

  void setoyako(GraphNode kid, int lcost) {

    //子供設定
    if(children == null){
      ListNode aln = new ListNode();
      aln.gnode = kid;
      aln.linkcost = lcost;
      children =aln;
    }else {
      ListNode org = children;
      ListNode nt = org.next;
      while(nt!=null){
org = nt;
nt = org.next;
      }
      ListNode aln = new ListNode();
      aln.gnode = kid;
      aln.linkcost = lcost;
      org.next = aln;
```

```
    }

    //親の設定
    ListNode pt = kid.parents;
    if(pt==null){
      ListNode aln = new ListNode();
      aln.gnode = this;
      aln.linkcost = lcost;
      kid.parents = aln;
    } else {
      ListNode org = pt;
      ListNode nt = org.next;
      while(nt!=null){
org = nt;
nt = org.next;
      }
      ListNode aln = new ListNode();
      aln.gnode = this;
      aln.linkcost = lcost;
      org.next = aln;
    }
  }

  void PrintForCheck() {
    System.out.println("------------");
    System.out.println("id is " + this.id);
    System.out.println("pos is " + this.pos);
    System.out.println("cost is " + this.cost);
    System.out.println("sumcost is "+this.sumcost);
    System.out.println("str is"+this.str);

    int cary[] = new int[20];
    int i = 0;
    ListNode a = this.children;
    while(a!=null){
      cary[i]=(a.gnode).id;i++;
      a = a.next;
    }
    int j;
```

```
    System.out.println("子供数は"+i+"個");
    for(j=0;j<i;j++){
      System.out.println("その id は"+cary[j]);
    }
    i = 0;
    a = this.parents;
    while(a!=null){
      cary[i] = (a.gnode).id; i++;
      a = a.next;
    }
    System.out.println("親の数は"+i+"個");
    for(j=0;j<i;j++){
      System.out.println("その id は"+cary[j]);
    }
  }
}

class ListNode {
  GraphNode gnode;
  int linkcost;
  ListNode next;
}
```