2018 年度 情報領域演習第三
— 第 15 回 —

注意事項

- 今回がいよいよ最終回である．これまでの演習で学習したさまざまな知識を活用して取り組む必要がある．
- 今回もグラフ構造データを扱うアルゴリズムについて実装を通じて学習する．アルゴリズムを理解しないと解くことが難しいため，いきなり問題文や入出力例を読まずに，必ずその前にある説明をよく読むこと．わからないことがあれば TA や教員に質問しよう．なお，一部の問題はこれまで作成した関数を利用する必要がある．
- 演習時間終了時点での提出状況についても評価の対象となるので，時間内にできるだけ多くの問題に挑戦するとよい．また，遅刻や途中退室は記録に残るので注意すること．
- プログラムの形式が指定されている場合には，それに従うこと．従わなくても「成功」と表示されることがあるが，得点にはならない．また，採点の際は `checker`とは異なる入出力例を用いて動作確認するので，「失敗」する反例に対する小手先の対策だけでは得点にならないことがあるので注意せよ．

はじめに

前回同様，CED において以下のコマンド (青字の部分) を実行することで出席を表明できる．ただし，N は所属するクラス名 1，2，3 で置き換え，N の前には空白を入れないこと．

```
[p1610999@blue00 ~]> /ced-home/staff/18jr3/15/checkerN
提出開始: 1 月 xx 日 14 時 40 分 0 秒
提出締切: 1 月 yy 日 14 時 39 分 0 秒
ユーザ: p1610999, 出席状況: 2019-01-xx-14-58
 問題:    結果 | 提出日時            | ハッシュ値                        |
    1:  未提出 |                     |                                  |
    2:  未提出 |                     |                                  |
                        ⋮
```

出席を表明するには，授業の開始から 30 分以内に実行する必要がある．実行しても出席状況が「欠席」である場合は教員に伝えること．なお，上の出力は例であるので，実際の締切りは各自コマンドで確認せよ．

提出も同じコマンドを用いて以下のように実行する．ただし，N はクラス名 (1 から 3)，X は 1 から 7 のいずれかの問題番号であり，`prog.c` は提出する C プログラムのファイル名であり，X の前後には忘れずに空白を入れること．

```
[p1610999@blue00 ~]> /ced-home/staff/18jr3/15/checkerN X prog.c
```

ここで，ファイル名として指定するのは，問題番号 X の問題を解く C プログラムのソースファイルであり，コンパイル済みの実行ファイルではない．ファイル名は特に指定しないが，「英数字からなる文字列 `.c`」などとすることが望ましい．このコマンドを実行することにより，指定されたファイルがコンパイルされ，実行テストプログラムが自動的に起動される．コンパイルに失敗した場合にはエラーとなり，提出されたことにはならないので注意すること．コンパイルが成功した場合には複数回の実行テストが行われ，実行テストにも成功すると「成功」と表示される．実行テストに失敗すると「失敗」と表示されるとともに反例となる入力と出力が表示されるので，失敗した理由を見つけるための参考にするとよい．なお，入出力が大きい場合やファイルとして入力したい場合には共に表示されるパスにあるファイルを使うと便利である．ただし，入力のない問題の場合は失敗しても反例は出力されない．

正しく提出できたか確認するためには以下のように出席の表明と同じコマンドを用いて確認できる．

```
[p1610999@blue00 ~]> /ced-home/staff/18jr3/15/checkerN
提出開始: 1 月 xx 日 14 時 40 分 0 秒
提出締切: 1 月 yy 日 14 時 39 分 0 秒
ユーザ: p1610999, 出席状況: 2019-01-xx-14-47
 問題:    結果 | 提出日時            | ハッシュ値                          |
    1:    成功 | 2019-01-xx-15-33   | 9b762c81932f3980cf03a768e044e65b |
                        ⋮
```

ハッシュ値は，提出した C プログラムのソースファイルの MD5 値である．CED では `md5sum` コマンドを用いて MD5 値を見ることにより，提出したファイルと手元のファイルが同じものであるかを確認することができる．

```
[p1610999@blue00 ~]> md5sum prog.c
9b762c81932f3980cf03a768e044e65b
```

なお，一度「成功」となった問題に対し，実行テストに失敗するプログラムを送信してしまうと，結果が「失敗」になってしまうので注意すること．

1 駅間の最短経路問題

前回に引き続き，今回もグラフ，特に路線図データを利用した問題を解いてみよう．今回の目標は最短経路問題を解くダイクストラ (Dijkstra) のアルゴリズムを実装することに加え，それを利用して路線図データから駅間の最短経路を求めるプログラムを作成することである．まず，ダイクストラのアルゴリズムを実装するために必要な集合の操作から実装してみよう．

1.1 集合の準備

ダイクストラのアルゴリズムでは,「グラフの頂点の集合」を効率よく管理することで実現される．今回扱うグラフは路線図データで頂点は駅番号なので，駅番号 (つまり `int` 型の値) の集合を考えればよい．集合として，次のような配列を使ったデータ構造を考えよう．

```
struct set { int elements[SETMAX]; int size; };
```

メンバ `size` は集合に含まれる要素の数，メンバ `elements` は集合の要素を格納する配列で，先頭 `size` 個 (0 番めから `size`−1 番めまで) に集合の要素が格納されている．`SETMAX` はマクロで定義されている整数値で，集合に格納できる要素の数の最大値である．たとえば，`SETMAX` の値が 9 であるような次の構造体 `set` を考えよう．

`size` = 6,　　`elements` =

0	1	2	3	4	5	6	7	8
6	3	1	7	8	4	0	2	5

メンバ `elements` の配列の要素は 9 個であるが，メンバ `size` が 6 で 0 番めから 5 番めまでが集合の要素であるため，この構造体が表す集合は $\{6, 3, 1, 7, 8, 4\}$ である．なお，今回扱う集合には 0 以上の駅番号を格納するため，集合の要素はすべて負でない整数であると仮定して実装する．また，後に示すプログラムでは，すべての駅番号を格納するために `SETMAX` を `10600` としている．

問題 1

この問題の目的は，与えられた条件を満たすように構造体 `set` を初期化する関数を作成することである．具体的には，次の関数 `init_set` を実装すればよい．

```
void init_set(struct set *p, int n, int e);
```

この関数は「`e` を除く 0 から `n`−1 までの整数を要素とする集合を，アドレス `p` の指す構造体に格納する関数」である．たとえば，`n` が 7，`e` が 4 なら，0 から 6 までの 7 個の整数のうち 4 を除いた集合 $\{0, 1, 2, 3, 5, 6\}$ を表す以下のような構造体をアドレス `p` に格納する．

size = 6, elements =

0	1	2	3	4	5	6	7	8
0	1	2	3	5	6	2	0	···

メンバ elements には要素が昇順に格納されているものとするが，n−1 番め以降の要素には何が格納されていても構わない．なお，この関数を呼ぶ時点で，アドレス p には構造体 set のためのメモリが確保されていると仮定してよい．

作成すべきプログラムの形式は後に示す main 関数を含むものとし，関数 init_set の定義を適切に埋めることによりプログラムを完成させよ．標準入力から与えられる入力は 1 行で，2 つの正の整数値 n, e が空白で区切られた文字列である (ただし，$e < n$)．標準出力には，$\{0, \ldots, n-1\}$ から e を除いた集合の要素を空白で区切った文字列を{と}で囲んで出力する (与えられる関数 print_set を用いればよい)．

入力例

```
7 4
```

出力例

```
{ 0 1 2 3 5 6 }
```

入力例

```
5 0
```

出力例

```
{ 1 2 3 4 }
```

入力例

```
8 7
```

出力例

```
{ 0 1 2 3 4 5 6 }
```

作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えればよいが，必要に応じて補助関数を定義してもよい．

```c
#include<stdio.h>
#define SETMAX 10600 /* 集合の要素数の最大値（駅の数）*/

struct set { int elements[SETMAX]; int size; };

void init_set(struct set *p, int n, int e) {
  /* ※ここを適切なプログラムで埋める */
}

void print_set(struct set *p) {
  int i;
  printf("{");
  for(i=0;i<p->size;++i) printf("␣%d",p->elements[i]);
  printf("␣}\n");
}

int main() {
  int n, e;
  struct set s;             /* 集合 s を用意 */
  scanf("%d␣%d␣", &n, &e); /* 入力から n と e を読み込み */
  init_set(&s, n, e);       /* s のアドレスと n と e を渡す */
  print_set(&s);
  return 0;
}
```

1.2 集合からの最小要素の取り出し

ダイクストラのアルゴリズムにおける集合の操作として，集合から最小値を取り除く，という操作がある．この操作を実現する関数を順を追って実装してみよう．

問題 2

この問題の目的は，構造体 set で与えられた集合に対し，最小値を取り出す関数 delete_min_int を作成することである．この問題で実装する関数 delete_min_int の引数と返り値の型は以下の通りとする．

```
int delete_min_int(struct set *p);
```

この関数は「アドレス p の指す構造体 set が表す集合から最も小さい要素を削除し，その要素を返す関数」である．集合には同じ要素が 2 回現れることはないと仮定してよい．また，与えられた集合が空集合であるとき (つまり，メンバ size が 0 であるとき) は -1 を返す．

関数 delete_min_int による操作は次の手順で実装することが求められる．たとえば，次のような構造体 set を考えよう．

size = 6, elements =

0	1	2	3	4	5	6	7	8
6	3	1	8	7	4	2	0	5

この構造体が表す集合は $\{6, 3, 1, 8, 7, 4\}$ である．この構造体に対して関数 delete_min_int を適用すると最小値である 1 が返されるが，集合からその要素が削除されなければならない．このため，集合に含まれる要素のうち最も後ろの 5 番めの要素を最小値の 1 のあった 2 番めに格納し，メンバ size を 1 つ減らせば，この構造体が表す集合は $\{6, 3, 4, 8, 7\}$ となり，最小値を取り出した集合が得られる (下図)．

size = 5, elements =

0	1	2	3	4	5	6	7	8
6	3	4	8	7	4	2	0	5

5 番めの要素は 4 のままであるが，メンバ size を 5 に変更しているため，集合の要素は 0 番めから 4 番めまでなので気にする必要はない．さらに，この構造体に対して関数 delete_min_int を適用すると最小値である 3 が返され，1 番めに最も後ろの 4 番めの要素を格納する (下図)．

size = 4, elements =

0	1	2	3	4	5	6	7	8
6	7	4	8	7	4	2	0	5

この結果，構造体 set の表す集合は $\{6, 7, 4, 8\}$ となる．一般的には，次の手順で最小値の削除が行われる．

1. メンバ size が 0 のときは，-1 を返す．
2. メンバ size が 0 でないときは，すべての要素 (つまり，elements の 0 番めから size−1 番めまで) を見て最も小さいものを探し，その添字を i とする．
3. 返り値として返すために i 番めの要素を記憶しておいてから，i 番めに size−1 番めの要素を格納する．
4. メンバ size を 1 つ減らし，前の手順で記憶していた最小値を返す．

作成すべきプログラムの形式は後に示す main 関数を含むものとし，関数 delete_min_int の定義を適切に埋めることによりプログラムを完成させよ．標準入力から与えられる入力は 100 行以下の複数行からなり，各行は正の整数値を表す文字列であり，これらを集合として構造体 set に格納したものを用意する (プログラムの形式通りに記述すればよい)．標準出力には，この構造体 set に対して delete_min_int を繰り返し適用した結果を毎回「取り出した最小値 : 残りの集合を問題 1 と同じ出力形式で表現したもの」の形で 1 行ずつ出力していき，集合が空になった時点で出力を終了する．

入力例

```
6
3
1
8
7
4
```

出力例

```
1 : { 6 3 4 8 7 }
3 : { 6 7 4 8 }
4 : { 6 7 8 }
6 : { 8 7 }
7 : { 8 }
8 : { }
```

入力例

```
5
4
3
2
1
```

出力例

```
1 : { 5 4 3 2 }
2 : { 5 4 3 }
3 : { 5 4 }
4 : { 5 }
5 : { }
```

入力例

```
1
2
3
4
5
```

出力例

```
1 : { 5 2 3 4 }
2 : { 5 4 3 }
3 : { 5 4 }
4 : { 5 }
5 : { }
```

作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えればよいが，必要に応じて補助関数を定義してもよい．

```c
#include<stdio.h>
#define SETMAX 10600 /* 集合の要素数の最大値（駅の数）*/
char buf[256];       /* 入力された文字列を格納するグローバル変数 */

struct set { int elements[SETMAX]; int size; };

int delete_min_int(struct set *p) {
  /* ※ここを適切なプログラムで埋める */
}

void print_set(struct set *p) {
  /* ※ここは問題 1 と同じ */
}

int main() {
  int i, m;
  struct set s; /* 入力される整数値を格納する集合 */
  i = 0;        /* 添字 0 から格納していく */
  while(fgets(buf,sizeof(buf),stdin)!=NULL) {
    sscanf(buf, "%d␣", &s.elements[i]); /* 入力された整数値を集合 s に格納 */
    ++i;                                /* 添字を 1 つ増やす */
  }
  s.size = i;   /* ← この時点で i に要素の数が格納されているはず */
  while(1) {
    m = delete_min_int(&s);
    if(m<0) break;                    /* m が負なら集合が空なので終了 */
    printf("%d␣:␣",m); print_set(&s); /* 最小値と集合を出力 */
  }
  return 0;
}
```

問題 3

この問題の目的も問題 2 と同じく，構造体 set で与えられた集合に対して「最小」のものを取り出す関数 delete_min を作成することであるが，ここでの最小値は単に数値として小さいということではなく，グローバル変数として与えられる配列 dist の値が最も小さくなるものを取り出さなければならない．すなわち，dist[m] が最小になるような m を集合から取り出す．具体的には，次の関数 delete_min を実装すればよい．

```c
int delete_min(struct set *p);
```

この関数は「アドレス p の指す構造体 set が表す集合の要素のうち，dist[m] の値が最も小さくなる要素 m を削除し，その要素を返す関数」である．与えられた集合が空集合であるとき，(つまり，メンバ size が 0 であるとき) は -1 を返す．なお，dist[m] を最小にするような m が複数ある場合にはメンバ elements における添字の小さい方を削除するものとする．

たとえば，配列 dist が次のように与えられているとしよう．

dist =

0	1	2	3	4	5	6	7	8	9
81	52	46	80	31	67	56	31	38	40

また，集合 $\{3,2,7,0,4,5\}$ を表す構造体 set が次のように与えられているとする．

size = 6, elements =

0	1	2	3	4	5	6	7	8
3	2	7	0	4	5	1	8	6

このとき，関数 delete_min によって取り出される「最小」の要素は，dist[3]，dist[2]，dist[7]，dist[0]，dist[4]，dist[5] の値を比較することで見つけることができる．最小のものは dist[7] と dist[4] でどちらも 31 であるが，「dist[m] を最小にするような m が複数ある場合にはメンバ elements における添字の小さい方」を選ぶので，2 番めの要素である 7 の方を選ぶ．この 7 を問題 2 と同様に削除するため，以下のような構造体 set に更新されることが期待される ($\{3,2,5,0,4\}$ を表す集合になる)．

size = 5, elements =

0	1	2	3	4	5	6	7	8
3	2	5	0	4	5	1	8	6

作成すべきプログラムの形式は後に示す main 関数を含むものとし，関数 delete_min の定義を適切に埋めることによりプログラムを完成させよ．標準入力から与えられる入力は複数行からなり，各行は 0 以上 SETMAX 未満の整数を表す文字列で，それらを要素とする集合を構造体 set の変数 s に格納する．標準出力には，この s に対して delete_min を繰り返し適用した結果を問題 2 と同じ要領で出力していき，集合が空になった時点で出力を終了する．最小の要素を見つけるために使用されるグローバル変数 dist はプログラム形式の中で与えられる．

入力例

```
3
2
7
0
4
5
```

出力例

```
7 : { 3 2 5 0 4 }
4 : { 3 2 5 0 }
2 : { 3 0 5 }
5 : { 3 0 }
3 : { 0 }
0 : { }
```

入力例

```
5
4
3
2
1
0
```

出力例

```
4 : { 5 0 3 2 1 }
2 : { 5 0 3 1 }
1 : { 5 0 3 }
5 : { 3 0 }
3 : { 0 }
0 : { }
```

入力例

```
0
1
2
3
4
5
```

出力例

```
4 : { 0 1 2 3 5 }
2 : { 0 1 5 3 }
1 : { 0 3 5 }
5 : { 0 3 }
3 : { 0 }
0 : { }
```

作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えればよいが，必要に応じて補助関数を定義してもよい．

```c
#include<stdio.h>
#define SETMAX 10600
char buf[256];      /* 入力された文字列を格納するグローバル変数 */
/* 最小要素を見つけるために利用される配列変数 dist (40 のあとの要素はすべて 0) */
int dist[SETMAX] = { 81, 52, 46, 80, 31, 67, 56, 31, 38, 40 };

struct set { int elements[SETMAX]; int size; };

int delete_min(struct set *p) {
  /* ※ここを適切なプログラムで埋める */
}

void print_set(struct set *p) {
  /* ※ここは問題 1 と同じ */
}

int main() {
  int i, m;
  struct set s; /* 入力される整数値を格納する集合 */
  i = 0;        /* 添字 0 から格納していく */
  while(fgets(buf,sizeof(buf),stdin)!=NULL) {
    sscanf(buf, "%d␣", &s.elements[i]); /* 入力された整数値を集合 s に格納 */
    ++i;                                /* 添字を 1 つ増やす */
  }
  /* この時点で i に要素の数が格納されているはず */
  s.size = i;
  while(1) {
    m = delete_min(&s);
    if(m<0) break;      /* m が負なら集合が空なので終了 */
    printf("%d␣:␣",m);  /* 見つけた最小要素を出力 */
    print_set(&s);
  }
  return 0;
}
```

1.3 ダイクストラのアルゴリズム

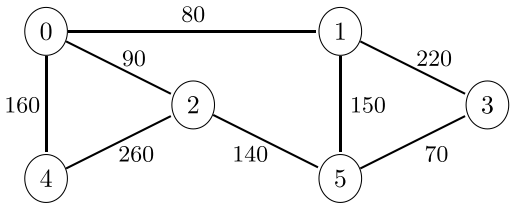
最初に述べたように，今回の演習の主な目的は「路線図データに対して，2 つの駅を結ぶ最短経路を見つけるプログラムを作成すること」である．この問題は，一般的にはグラフに対する**最短経路問題**と呼ばれ，最も有名なアルゴリズムがダイクストラのアルゴリズムである．ここまでの頂点の集合の扱いさえきちんと理解できていれば，このアルゴリズムの実装することはそれほど難しくない．とはいえ「どこを通れば最短でたどり着けるか」を求めるのはちょっとだけ複雑なので，まず,「最短でたどり着いたとしたらその距離はいくらか」を求めるプログラムを作成してから，経路を求めるプログラムを作成する．なお，前回の演習で実装したワーシャルのアルゴリズムを発展させた「ワーシャル・フロイドのアルゴリズム [1]」も最短経路問題を解くアルゴリズムであるが，あらゆる 2 つの点の間の最短経路を見つけることが目的のため，今回のように巨大なグラフを扱う問題には適していない [2]．

前回に引き続き，路線図データを表すグラフは次の形式を用いる．

[1] ワーシャルもフロイドも人名であるが，英語では Floyd-Warshall algorithm と逆の順序で呼ばれることが多い．

[2] 単一始点最短経路 (Single-Source Shortest Path, SSSP) 問題を解くのがダイクストラのアルゴリズムであり，全点対間最短経路 (All-Pairs Shortest Path, APSP) 問題を解くのがワーシャル・フロイドのアルゴリズムである．

```
6
0:1:1013:80
0:2:1012:90
0:4:1011:160
1:3:1013:220
1:5:1015:150
2:5:1012:140
2:4:1014:260
3:5:1012:70
```



前回と異なり，距離を `float`型 (単位:km) ではなく `int`型 (単位:m) としているので，プログラムを再利用する場合は注意しよう．ダイクストラのアルゴリズムでは，グラフを隣接リストで表現する必要があるので，前回の演習で利用した構造体 `node` を用いる (ただし，`kyori` は `int` 型に変更されている)．

```
struct node { int eki, rosen, kyori; struct node *next; };
```

メンバ `eki` は駅番号，メンバ `rosen` は路線番号，メンバ `kyori` は距離，メンバ `next` は次の節点のアドレスを表す．連結リストの終わりであれば `next` には `NULL` を入れる．グラフの隣接リスト表現は，構造体 `node` のアドレスを要素とする配列で表される．前回 (第 14 回) の問題 1 で作成した関数 `add_edge` を用いればよいが，以下のように引数 `kyori` の型を変更する必要がある．

```
void add_edge(struct node *adjlist[], int eki1, int eki2, int rosen, int kyori);
```

問題 4

この問題の目的は，隣接リスト表現で与えられた路線図データに対し，2 つの駅の間の最短経路の距離 (最短距離) を求める関数を作成することである．具体的には，次の関数 `dijkstra` を実装すればよい．

```
int dijkstra(struct node *adjlist[], int eki1, int eki2, int ekisu);
```

この関数は「頂点数 `ekisu` のグラフを表す隣接リスト表現 `adjlist` に対して，駅 `eki1` から駅 `eki2` までの最短距離を返す関数」である．駅 `eki1` から駅 `eki2` に到達可能でなければ，`INT_MAX` (`int` 型の最大値)[3] を返す．

駅 `eki1` と駅 `eki2` の間の最短距離はダイクストラのアルゴリズムによって求めることができる．ダイクストラのアルゴリズムは，1 つの頂点から他のすべての頂点への最短距離を計算するためのアルゴリズムであるため，駅 `eki1` から他のすべての駅への最短距離を計算することで，駅 `eki1` から駅 `eki2` への最短距離を求めることができる．ダイクストラのアルゴリズムの基本的なアイデアは，「近い方の頂点から順に最短距離を確定させていく」というもので，駅 `eki1` からの最短距離を格納する配列 `dist` をグローバル変数として用意することで実装できる (駅 `eki1` から駅 i までの最短距離が `dist[`i`]` に格納される)．

1. 配列 `dist` の `eki1` 番めの要素を 0 とし，それ以外の要素を `INT_MAX` にしておく (駅 `eki1` から駅 `eki1` までの最短距離は 0 で確定，それ以外はまだ到達可能かわからないので最大の整数値としておく)．
2. 直前に最短距離を確定した駅を格納する変数 `cur` に `eki1` を入れ，まだ最短距離を確定していない駅の集合 `unknown` として，「0 から `ekisu`−1 までのうち `eki1` を除くすべての整数が入った集合」を用意する．
3. `unknown` が空になるか `cur` が `eki2` と等しくなるまで次を繰り返す[4]．
   i. 駅 `cur` に隣接するすべての駅について次を実行する：
   その駅への最短距離候補 (`dist` に記録されている) よりも，「駅 `cur` までの最短距離」と「駅 `cur` とその駅の間の距離」の和 (つまり，駅 `cur` を経由してその駅に着く場合の距離) の方が小さければ，その距離を最短距離として新たに `dist` に登録する．

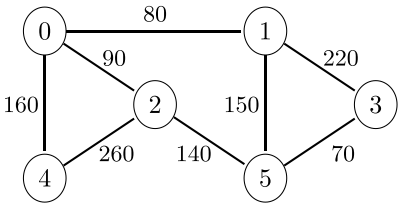
[3] この定数を使うためにはヘッダファイル `limit.h` をインクルードする必要がある．
[4] ダイクストラのアルゴリズムでは，1 つの頂点からすべての頂点までの最短経路が計算されるが，この関数では `eki2` までの最短距離さえ確定すれば終了してよいため，「`cur` が `eki2` と等しくなる」という条件も繰り返しの終了条件に追加されている．

ii. 「集合 unknown に含まれる駅のうち，最短距離が最も小さいもの」を取り出し，それを最短距離を確定した駅として cur に入れる (最短距離がまだ確定していない駅のうち，最も eki1 に近い駅を確定).

4. eki2 までの最短距離として dist[eki2] を返す.

この手続きのうち,「0 から ekisu−1 までのうち eki1 を除くすべての整数が入った集合」を unknown に格納する部分では問題 1 で定義した init_set を用い,「集合 unknown に含まれる駅のうち，最短距離が最も小さいもの」を取り出す部分では問題 3 で定義した delete_min を用いればよい.

たとえば，右のようなグラフが与えられ，駅 4 から駅 5 までの最短距離を求めたい場合を考えよう．ステップ 1 により，dist の要素は以下のようになる (ここでは INT_MAX を ∞ で表している).



	0	1	2	3	4	5
dist =	∞	∞	∞	∞	0	∞

続いてステップ 2 により，最短距離を確定した駅 cur を 4 とし，まだ最短距離を確定していない駅の集合 unknown をそれ以外の駅の集合 $\{0,1,2,3,5\}$ とする．さらに，ステップ 3 の繰り返しは次のように進められる.

(I) cur である 4 に隣接する駅 0，駅 2 について，その駅までの現在の最短距離 (dist にある値) と，駅 4 を経由してその駅に着いたときの距離 (dist[4]+d, d は駅 4 とその駅の間の距離) を比較する．いまの場合，現在の最短距離はどれも ∞ なので，どちらも駅 4 を経由してその駅に着いたときの距離で更新されるため，dist[0] を dist[4]+160(= 160)，dist[2] を dist[4]+260(= 260) とする.

	0	1	2	3	4	5
dist =	160	∞	260	∞	0	∞

この結果，集合 unknown= $\{0,1,2,3,5\}$ に含まれる駅のうち，最も近い (dist の値が最も小さい) 駅 0 への最短距離が確定して cur に 0 が格納され，まだ確定していない駅の集合 unknown は $\{5,1,2,3\}$ となる.

(II) cur である 0 に隣接する駅 1，駅 2，駅 4 について，その駅までの現在の最短距離 (dist にある値) と，駅 0 を経由してその駅に着いたときの距離 (dist[0]+d, d は駅 0 とその駅の間の距離) を比較する．駅 1 については現在最短距離が ∞ なので，dist[0]+80(= 240) の方が小さいため dist[1] を 240 とする．駅 2 については現在 260 が最短距離であるが，駅 0 を経由した場合の距離 dist[0]+90(= 250) の方が小さいため dist[2] を 250 とする．駅 4 については dist[4]= 0 なので，駅 0 を経由した方が遠いため更新されない.

	0	1	2	3	4	5
dist =	160	240	250	∞	0	∞

この結果，集合 unknown= $\{5,1,2,3\}$ に含まれる駅のうち，最も近い (dist の値が最も小さい) 駅 1 の最短距離が確定して cur に 1 が格納され，unknown は $\{5,3,2\}$ となる.

(III) cur である 1 に隣接する駅 0, 駅 3, 駅 5 について，その駅までの現在の最短距離 (dist にある値) と，駅 1 を経由してその駅に着いたときの距離 (dist[1]+d, d は駅 1 とその駅の間の距離) を比較する．駅 0 については駅 1 を経由した方が遠くなるので最短距離は変わらないが，駅 3 と駅 5 についてはまだ最短距離が ∞ で駅 1 を経由した方が近いため，dist[3] を dist[1]+220(= 460)，dist[5] を dist[1]+150(= 390) とする.

	0	1	2	3	4	5
dist =	160	240	250	460	0	390

この結果，集合 unknown= $\{5,2,3\}$ に含まれる駅のうち，最も近い (dist の値が最も小さい) 駅 2 への最短距離が確定して cur に 2 が格納され，unknown は $\{5,3\}$ となる.

(IV) cur である 2 に隣接する駅 0，駅 4，駅 5 について，その駅までの現在の最短距離 (dist にある値) と，駅 2 を経由してその駅に着いたときの距離 (dist[2]+d, d は駅 2 とその駅の間の距離) を比較する．駅 0，駅 4 については駅 2 を経由した方が遠くなるので最短距離は変わらない．駅 5 についても駅 2 を経由したても距離は同じ (dist[2]+140 = 390) なので dist は更新されない．

dist =

0	1	2	3	4	5
160	240	250	460	0	390

この結果，集合 unknown= $\{5,3\}$ に含まれる駅のうち，最も近い (dist の値が最も小さい) 駅 5 への最短距離が確定して 5 が cur に格納され，unknown は $\{3\}$ となる．ここで目的地である駅 5 が cur に格納されたので手続きは終了し，ステップ 4 で dist[5] の 390 が返される．

作成すべきプログラムの形式は後に示す main 関数を含むものとし，関数 dijkstra の定義を適切に埋めることによりプログラムを完成させよ．このプログラムでは，路線図データをファイルから入力するように記述されているため，マクロで ROSENZU として指定されているファイル rosenzu.txtを同じディレクトリに置く必要がある．ファイル rosenzu.txt は講義ページからダウンロードすること．標準入力から与えられる入力は 1 行で，2 つの駅の駅番号を空白で区切った文字列が与えられる．標準出力にはそれらの駅間の最短距離を出力する．なお，このプログラムの動作確認を行う際にはいきなり大きな路線図データで試すとバグが見つかりにくいので，より小さな路線図データ rosenzu-s.txt (上の例で示したグラフと同じもの) も用意した．ファイル rosenzu-s.txt も講義ページからダウンロードできる．以下の 3 つは，ROSENZUを rosenzu-s.txtとした場合の入出力例である．後に示すプログラムの形式のマクロを適切に書き換えて試すこと．

入力例	出力例
4 1	240

入力例	出力例
0 3	300

入力例	出力例
4 5	390

次の 3 つは，ROSENZUを rosenzu.txtとした場合の入出力例である．つまり，後に示すプログラムの形式のマクロを変更せずに実行したものである．

入力例	出力例
5430 5413	4401

入力例	出力例
7446 5411	22510

入力例	出力例
8098 10112	956973

作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えればよいが，必要に応じて補助関数を定義してもよい．

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<limits.h>
4 #define ROSENZU "rosenzu.txt" /* 路線図データファイル */
5 #define SETMAX 10600 /* 集合の要素数の最大値（駅の数）*/
6 char buf[256];       /* 入力された文字列を格納するグローバル変数 */
```

```
7  int dist[SETMAX];    /* 指定された駅から各駅までの最短距離を格納するグローバル変数 */
8
9  struct node { int eki, rosen, kyori; struct node *next; };
10 struct set { int elements[SETMAX]; int size; };
11
12 void init_set(struct set *p, int n, int e) {
13   /* ※ここは問題 1 と同じ */
14 }
15
16 int delete_min(struct set *p) {
17   /* ※ここは問題 3 と同じ */
18 }
19
20 void add_edge(struct node *adjlist[], int eki1, int eki2, int rosen, int kyori) {
21   /* ※ここは第 14回の問題 1と同じ．ただし,kyori の型は int 型 */
22 }
23
24 int dijkstra(struct node *adjlist[], int eki1, int eki2, int ekisu) {
25   /* ※ここを適切なプログラムで埋める */
26 }
27
28 int main() {
29   int eki1, eki2, rosen, ekisu, i, kyori;
30   FILE *fp = fopen(ROSENZU,"r");   /*  路線図ファイルを読む準備 */
31   fscanf(fp, "%d␣", &ekisu);       /* 1行めの駅数を読取り */
32   struct node *adjlist[ekisu];
33   /* 隣接リスト表現を初期化．すべての頂点に対する隣接リストを空にする */
34   for(i=0;i<ekisu;++i) adjlist[i] = NULL;
35   while(fgets(buf,sizeof(buf),fp)!=NULL) {
36     /* 隣り合う駅の情報を読取り */
37     sscanf(buf, "%d:%d:%d:%d␣", &eki1, &eki2, &rosen, &kyori);
38     /* そのデータを隣接リスト表現のグラフに追加 */
39     add_edge(adjlist, eki1, eki2, rosen, kyori);
40   }
41   fclose(fp);
42   scanf("%d␣%d␣", &eki1, &eki2);
43   printf("%d\n", dijkstra(adjlist, eki1, eki2, ekisu));
44   /* for(i=0;i<ekisu;++i)
45     if(dist[i] < INT_MAX)
46       printf("%d: %d\n", i, dist[i]); */
47   return 0;
48 }
```

次に最短距離だけでなく最短経路を求めるプログラムを作成しよう．基本的なアイデアは最短距離を求める場合とほぼ同じであるが，最短距離だけでなく「最短になる場合の 1 つ前の駅」も保持する点が異なる．

問題 5

隣接リスト表現で与えられた路線図データに対し，2 つの駅の間の最短経路の距離 (最短距離) を求める関数を作成することである．具体的には，次の関数 `dijkstra_path` を実装すればよい．

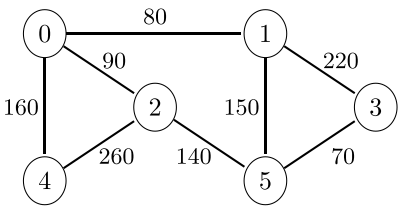
```
int dijkstra_path(struct node *adjlist[], int eki1, int eki2, int ekisu);
```

この関数は問題 4 の関数 `dijkstra` と同様に，「頂点数 `ekisu` のグラフを表す隣接リスト表現 `adjlist` に対して，駅 `eki1` から駅 `eki2` までの最短距離を返す関数」であり，駅 `eki1` から駅 `eki2` に到達可能でなければ，`INT_MAX` (`int` 型の最大値) を返す．関数 `dijkstra` と異なる点は，最短距離の計算の過程で 2 つの配列 `prev`

と hop の内容を更新する点で，これらの配列は最短距離である経路 (最短経路) を求めるために使用される．これらの配列をどのように使用するかを説明する前に，本問題における「最短経路」を厳密に定義しておく．

まず，駅 eki1 から駅 eki2 までの最短距離といえば 1 つに定まることは明らかだろう．「2 つの駅間の最短距離が 100 でも 200 でもある」などということはあり得ず，小さい方が最短距離になる．一方，最短経路についてはどうだろうか．

たとえば，右のグラフの場合，0 から 3 への最短距離は 300 であるが，$0 \to 1 \to 3$ も $0 \to 1 \to 5 \to 3$ も $0 \to 2 \to 5 \to 3$ も距離 300 の最短経路である．5 から 4 への最短距離は 390 であるが，$5 \to 1 \to 0 \to 4$ も $5 \to 2 \to 0 \to 4$ も距離 390 の最短経路である．このように最短経路の場合は 1 つに定まるとは限らない．そこで，今回の問題では以下の規準によって最良の最短経路を定める．



- 最短経路のうち最も通る駅が少ない経路を最良とする．
- 通る駅が少ない経路が複数ある場合には，目的駅の手前の駅の駅番号が最も小さい経路を最良とする．それも複数ある場合にはさらに手前の駅のたどって比較していく．

先ほどの例で言えば，0 から 3 への最良の最短経路は最も通る駅の少ない $0 \to 1 \to 3$ となる．5 から 4 への最短経路については $5 \to 1 \to 0 \to 4$ と $5 \to 2 \to 0 \to 4$ を比較して，目的駅 4 の手前の駅はどちらも 0 であるが，その手前の駅は 1 と 2 なので，$5 \to 1 \to 0 \to 4$ が最良の最短経路となる．

先述の通り，これらの最短経路は問題 4 のプログラムに少し手を加えるだけで求めることができる．最短経路になる場合の駅の数を保持する配列 hop と最短経路になる場合の手前の駅を保持する配列 prev を用意し，配列 dist を更新すると同時にこれらを適切に更新すればよい．より具体的に言えば，次の 4 つの点に注意して問題 4 のプログラムに手を加える．

- ステップ 1 で dist を初期化すると同時に，hop を初期化する．配列 hop は eki1 のみ 0 (0 ホップでたどり着くため)，その他はまだたどり着くかもわからないので INT_MAX とする．
- ステップ 3 の i で dist の値を更新するときは，hop も prev も更新する．たとえば，dist[eki] を dist[cur] + kyori で更新するなら，hop[eki] は hop[cur] より 1 駅多くなるようにし，prev[eki] には手前の駅の駅番号 cur を入れる．
- ステップ 3 の i で dist[eki] の値と経由した場合の距離 dist[cur]+kyori を比較するときの更新する規準を変更する． 変更される場合は以下の通り．
  - dist[eki] の方が大きいとき (最短になる経路を選ぶため)
  - dist[eki] と dist[cur]+kyori が等しいが，hop[eki] が hop[cur]+1 より大きいとき (通る駅が少ない経路を選ぶため)
  - dist[eki] と dist[cur]+kyori が等しく，hop[eki] と hop[cur]+1 も等しいが，prev[eki] が cur より大きいとき (手前の駅の駅番号が小さい経路を選ぶため)
- ステップ 3 の ii で集合 unknown に含まれる駅番号から「最も近い駅」を取り出す規準を変更する． 新しい規準は以下の通り．
  - dist の値が最小のもの (最短になる経路を選ぶため)
  - それが複数ある場合にはその中で hop の値が最小のもの (通る駅が少ない経路を選ぶため)
  - それも複数ある場合にはその中で prev の値が最小のもの (手前の駅の駅番号が小さい経路を選ぶため)

以上の点を変更すれば，cur に eki2 が格納された時点で，eki2 より近いすべての駅について，prev に「最短経路における 1 つ手前の駅の駅番号」が格納されているので，それを hop[eki2] の数だけたどっていけば eki1 にたどり着き，(逆向きの) 最短経路がわかる．

作成すべきプログラムの形式は後に示す main 関数を含むものとし，関数 dijkstra_path の定義を適切に埋めることによりプログラムを完成させよ．標準入力から与えられる入力は問題 4 と同様に 2 つの駅の駅

番号を空白で繋げた文字列で，路線図データのファイル名はマクロの ROSENZU に与えられるものとする (プログラム形式では rosenzu.txt になっているため，デバッグの際には rosenzu-s.txt に変更して試すとよい)．標準出力には，最短距離と最短経路を「最短距離 : 目的駅 <- 手前の駅 <- さらに手前の駅 <- ··· <- 出発駅」の形式で出力する．

以下の 3 つは，ROSENZUを rosenzu-s.txtとした場合の入出力例である．後に示すプログラムの形式のマクロを適切に書き換えて試すこと．

入力例

```
4 1
```

出力例

```
240: 1 <- 0 <- 4
```

入力例

```
0 3
```

出力例

```
300: 3 <- 1 <- 0
```

入力例

```
5 4
```

出力例

```
390: 4 <- 0 <- 1 <- 5 ← 4 <- 0 <- 2 <- 5 ではない
```

次の 3 つは，ROSENZUを rosenzu.txtとした場合の入出力例である．つまり，後に示すプログラムの形式のマクロを変更せずに実行したものである．

入力例

```
5430 5413
```

出力例

```
4401: 5413 <- 5412 <- 5411 <- 5428 <- 5429 <- 5430
```

入力例

```
5411 1230
```

出力例

```
15314: 1230 <- 5451 <- 5453 <- 5454 <- 5455 <- 5457
<- 5458 <- 5459 <- 5460 <- 5399 <- 5400 <- 5401
<- 5402 <- 5403 <- 5404 <- 5405 <- 5406 <- 5407
<- 5408 <- 5409 <- 5410 <- 5411
↑すべて 1 行で出力される
```

入力例

```
7446 5411
```

出力例

```
22510: 5411 <- 5410 <- 5409 <- 5408 <- 5407 <- 5406
<- 5405 <- 5404 <- 5403 <- 5402 <- 5401 <- 5400
<- 5399 <- 5398 <- 5397 <- 5396 <- 5395 <- 5470
<- 5452 <- 7461 <- 7462 <- 7463 <- 7464 <- 5859
<- 5858 <- 7382 <- 7383 <- 7445 <- 7446
↑すべて 1 行で出力される
```

作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えればよいが，必要に応じて補助関数を定義してもよい．

```c
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
#define ROSENZU "rosenzu.txt" /* 路線図データファイル */
#define SETMAX 10600 /* 集合の要素数の最大値（駅の数）*/
char buf[256];       /* 入力された文字列を格納するグローバル変数 */
int dist[SETMAX];    /* 指定された駅から各駅までの最短距離を格納するグローバル変数  */
int prev[SETMAX];    /* 各駅までの最短経路における 1つ前の駅を格納するグローバル変数 */
int hop[SETMAX];     /* 各駅までの最短経路における駅の数を格納するグローバル変数    */

struct node { int eki, rosen, kyori; struct node *next; };
struct set { int elements[SETMAX]; int size; };
```

```
13
14 void init_set(struct set *p, int n, int e) {
15   /* ※ここは問題 1 と同じ */
16 }
17
18 int delete_min(struct set *p) {
19   /* ※ここは問題 3 から変更する必要あり */
20 }
21
22 void add_edge(struct node *adjlist[], int eki1, int eki2, int rosen, int kyori) {
23   /* ※ここは問題 4 と同じ */
24 }
25
26 int dijkstra_path(struct node *adjlist[], int eki1, int eki2, int ekisu) {
27   /* ※ここを適切なプログラムで埋める */
28   /*   問題 4 の関数 dijkstra を適切に変更する */
29 }
30
31 int main() {
32   int eki, eki1, eki2, rosen, ekisu, i, kyori;
33   FILE *fp = fopen(ROSENZU,"r");   /* 路線図ファイルの読取り開始 */
34   fscanf(fp, "%d␣", &ekisu);       /* 1行めの駅数を読取り */
35   struct node *adjlist[ekisu];
36   /* 隣接リスト表現を初期化. すべての頂点に対する隣接リストを空にする */
37   for(i=0;i<ekisu;++i) adjlist[i] = NULL;
38   while(fgets(buf,sizeof(buf),fp)!=NULL) {
39     /* 隣り合う駅の情報を読取り */
40     sscanf(buf, "%d:%d:%d:%d␣", &eki1, &eki2, &rosen, &kyori);
41     /* そのデータを隣接リスト表現のグラフに追加 */
42     add_edge(adjlist, eki1, eki2, rosen, kyori);
43   }
44   fclose(fp);   /* 路線図ファイルの読取り終了 */
45   scanf("%d␣%d␣", &eki1, &eki2);
46   kyori = dijkstra_path(adjlist, eki1, eki2, ekisu);
47   /* 最短距離を出力 */
48   printf("%d:", kyori);
49   eki = eki2;
50   /* hop[eki]の数だけ前に戻って出力 */
51   for(i=0;i<hop[eki2];++i) {
52     printf("␣%d␣<-", eki); /* 1つ手前の駅を出力   */
53     eki = prev[eki];       /* さらに 1つ手前の駅へ */
54   }
55   /* プログラムが正しければ,ここで eki が開始駅 eki1 なっているはず (違っていたら異常終了) */
56   if(eki!=eki1) { fprintf(stderr,"hop␣or␣prev␣is␣invalid.\n"); exit(1); }
57   printf("␣%d\n", eki);    /* 開始駅を出力 */
58   return 0;
59 }
```

メモ　冒頭でも触れたようにダイクストラのアルゴリズムは，今回作成したプログラムよりももう少し効率化できる．特に「集合 unknown から dist の小さいものを選ぶ」という操作は，dist が小さいものが取り出しやすいように unknown を表す適切なデータ構造を用意しておけばよい．二分木ヒープやフィボナッチヒープと呼ばれるデータ構造を用いることが多いが，興味があれば各自で学習するとよい．

1.4 おまけ：駅名と駅番号の対応の探索

ここまでの問題が解けていれば，(新幹線の駅を除く) 日本全国の駅間の最短経路問題を解くことができたことになる[5]．しかしながら，探したい経路があっても駅名に対応する駅番号がわからないと検索できないし，検索された経路を見ても駅番号だけではどの駅を通っているのかわからない．そこで，以下の問題を通して駅名から駅番号へ変換するプログラムを作っておこう．なお，ここからの問題は日本語文字列の入出力を伴うが，CED の標準的な環境に合わせて文字コードは UTF-8 としている．他の環境で問題 6 と問題 7 に取り組む場合は文字コードの設定に注意が必要である．

問題 6

この問題の目的は駅名から駅番号を求めるプログラムを作成することである．同じ駅名でも全く違う場所にある駅で場合もあるし，路線図データでは路線が違えば違う駅番号を持つので，駅名から駅番号は 1 つに定まらない．たとえば，府中駅は「JR 福塩線」「よしの川ブルーライン」「京王線」の 3 つの路線にあるし，新宿駅は「JR 山手線」「JR 中央本線 (東京〜塩尻)」「JR 中央線 (快速)」「JR 中央・総武線」「JR 埼京線」「JR 成田エクスプレス」「JR 湘南新宿ライン」「京王線」「小田急線」「東京メトロ丸ノ内線」「都営大江戸線」「都営新宿線」の 12 の路線にある (これらは近いので路線図データ上は「徒歩」という路線で繋がれている)．駅名と対応する駅番号と路線名を並べたファイル `ekisen.txt` を用意したので，それをダウンロードしてこの問題を解く必要がある．なお，このファイルは駅名で昇順に並んでいるので，二分探索によって `strcmp` 関数[6] で比較しながら目的の駅を探せば効率がよい．ただし，その駅名に対応する駅番号と路線をすべて列挙する必要があるので，二分探索で見つけた後も適切に探索を続けなければならない．

作成すべきプログラムの形式は後に示す `main` 関数を含むものとし，※印のコメント部分を適切に埋めることでプログラムを完成させよ．標準入力からの入力は駅名を表す文字列で，標準出力にはその駅名に対応するすべての駅番号と路線名の組を:で区切り，(駅番号について昇順になるように)1 行に 1 組ずつ出力する．

入力例

```
調布
```

出力例

```
5411:京王線
5428:京王相模原線
```

入力例

```
府中
```

出力例

```
4014:JR 福塩線
4183:よしの川ブルーライン
5417:京王線
```

入力例

```
新宿
```

出力例

```
1233:JR 山手線
1395:JR 中央本線(東京〜塩尻)
1450:JR 中央線(快速)
1479:JR 中央・総武線
1721:JR 埼京線
1883:JR 成田エクスプレス
1963:JR 湘南新宿ライン
5394:京王線
5472:小田急線
5771:東京メトロ丸ノ内線
7403:都営大江戸線
7461:都営新宿線
```

作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えればよいが，必要に

[5] ただし，演習で用意した路線図データにおける駅間の距離は各駅の緯度と経度に基づいて計算されたものであるため正確ではない．

[6] 文字列を比較する関数．文字列 `s1` と文字列 `s2` に対して，`strcmp(s1,s2)` は `s1` が大きければ正の整数，`s2` が大きければ負の整数，互いに等しければ `0` を返す．

応じて補助関数を定義してもよい.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define SETMAX 10600
char buf[256];

struct station { int eki; char name[64], rosen[64]; };

int main() {
  int i=0, ekisize, l, r, m;
  struct station *ekidata[SETMAX], *st;
  char ekiname[64];
  FILE *fp = fopen("./ekisen.txt","r");
  while(fgets(buf,sizeof(buf),fp)!=NULL && i<SETMAX) {
    st = (struct station*) malloc(sizeof(struct station));
    sscanf(buf, "%[^:]:%d:%[^\n]*c", st->name, &st->eki, st->rosen);
    ekidata[i] = st;
    ++i;
  }
  fclose(fp);
  ekisize = i;
  scanf("%[^\n]%*c",ekiname);
  l=0; r=ekisize-1;
  /* ここから二分探索 */
  while(l<=r) {
    /* ※ここを適切なプログラムで埋める */
  }
  if(r<l) { /* 駅名がない場合 */
    fprintf(stderr, "%s:␣駅名がありません\n", ekiname); exit(1);
  } else { /* 駅名がある場合 */
    /* ※ここも適切なプログラムで埋める */
  }
  return 0;
}
```

問題 7

上のプログラムを使えば，調布から渋谷までの経路を検索したいとき，まず問題 6 のプログラムで,「調布」という入力で得られる出力から「5411:京王線」を選び,「渋谷」という入力で得られる出力から「1230:JR 山手線」を選び，この 5411 と 1230 を使って問題 5 のプログラムで経路を求めればよい．しかしながら，得られる結果も駅番号の表記であるために実際にどのような経路なのかがわからない．そこで，問題 5 を次のように書き換えて，駅名が表示されるように修正しよう．

- `scanf("%d␣%d", ...)` の前に駅名ファイル `ekisen.txt` を読み取るプログラム片を追加する．

```
char *ekiname[SETMAX], *senname[SETMAX], *ename, *sname;
fp = fopen("./ekisen.txt","r");
while(fgets(buf,sizeof(buf),fp)!=NULL) {
  ename = (char*) malloc(sizeof(char)*64);
  sname = (char*) malloc(sizeof(char)*128);
  sscanf(buf, "%[^:]:%d:%[^\n]*c", ename, &eki, sname);
  ekiname[eki] = ename;
  senname[eki] = sname;
}
fclose(fp);
```

- 出力部分を駅名と路線名がわかるように書き換える．

```
printf("␣%s␣(%s)␣<-", ekiname[eki], senname[eki]); /* for 文内 */

printf("␣%s␣(%s)\n", ekiname[eki], senname[eki]); /* 出発駅の出力 */
```

この問題は問題 5 さえできれば解けるサービス問題のため，入出力例のみを示す．

入力例

```
5430 5413
```

出力例

```
4401: 飛田給（京王線）<- 西調布（京王線）
<- 調布（京王線）<- 調布（京王相模原線）
<- 京王多摩川（京王相模原線）<- 京王稲田堤（京王相模原線）
↑すべて 1 行で出力される
```

入力例

```
5411 1230
```

出力例

```
15314: 渋谷（JR 山手線）<- 渋谷（京王井の頭線）
<- 神泉（京王井の頭線）<- 駒場東大前（京王井の頭線）
<- 池ノ上（京王井の頭線）<- 下北沢（京王井の頭線）
<- 新代田（京王井の頭線）<- 東松原（京王井の頭線）
<- 明大前（京王井の頭線）<- 明大前（京王線）
<- 下高井戸（京王線）<- 桜上水（京王線）
<- 上北沢（京王線）<- 八幡山（京王線）
<- 芦花公園（京王線）<- 千歳烏山（京王線）
<- 仙川（京王線）<- つつじヶ丘（京王線）
<- 柴崎（京王線）<- 国領（京王線）
<- 布田（京王線）<- 調布（京王線）
↑すべて 1 行で出力される
```

入力例

```
6837 3385
```

出力例

```
14828: 伊丹（JR 宝塚線）<- 猪名寺（JR 宝塚線）
<- 塚口（JR 宝塚線）<- 尼崎（JR 宝塚線）
<- 大阪（JR 宝塚線）<- 梅田（阪急京都本線）
<- 十三（阪急京都本線）
↑すべて 1 行で出力される
```