EDAF05 Exam

7 January 2010, 14.00–19.00

Thore Husfeldt, Computer Science, Lund University

Instructions

What to bring. You can bring any written aid you want. This includes the course book and a dictionary. In fact, these two things are the only aids that make sense, so I recommend you bring them and only them. But if you want to bring other books, notes, print-out of code, old exams, or today's newspaper you can do so. (It won't help.)

You can't bring electronic aids (such as a laptop) or communication devices (such as a mobile phone). If you really want, you can bring an old-fashioned pocket calculator (not one that solves recurrence relations), but I can't see how that would be of any use to you.

Filling out the exam Most questions are multiple choice. Mark the box or boxes with a cross or a check-mark. If you change your mind, completely black out the box and write your answer's letter(s) in the left margin. In case it's unclear what you mean, I will choose the least favourable interpretation.

In those questions where you have to write or draw something, I will be extremely unco-operative in interpreting your handwriting. So *write clearly*. Use English or Swedish. If there is a way to misunderstand what you mean, I will use it.

Scoring Each multiple choice question has exactly *one* correct answer. To get the maximum score for that question, you must check that answer, and only that. However, to reflect partial knowledge, you *may* check several boxes, in case you're not quite sure (this lowers your score, of course – the more boxes you check, the fewer points you score). If you check *no* boxes or *all* boxes, your score for that question is 0. If the correct answer is not among the boxes you checked, you score is negative, so it's better to *not* answer a question where you're on very thin ice. The worst thing you can do is to check all boxes except the correct one, which gives you a large negative score.

Want an example? Assume a question worth maximum 2 points has $k = 4$ possible answers (one of them correct).

- If you select only the correct answer, you receive 2 points.
- If you select 2 answers, one of which is correct, you receive 1 point.
- If you select 3 answers, one of which is correct, you receive 0.41 points.
- if you select no answer or all answers, you receive 0 point.
- If you select only one answer, and it is wrong, you receive -0.67 points.
- If you select 2 answers that are both wrong, you receive -1 point.
- If you select 3 answers that are all wrong, you receive -1.25 points.

As a special case, for a yes/no question, you receive 1, 0, or -1 points, depending on whether you answer is correct, empty, or wrong.

If you want to know the precise formula, if the question has k choices, and you checked a boxes, your score is $\log(k/a)$, provided you checked the correct answer, and $-a\log(k/a)/(k-a)$ if you only checked wrong answers. Moreover, I have weighted the questions by relevance (not necessarily difficulty), and indicated the maximum points with each question.

You really care why this scoring system makes sense? Then read [Gudmund Skovbjerg Frandsen, Michael I. Schwartzbach: A singular choice for multiple choice. SIGCSE Bulletin 38(4): 34–38 (2006)]. For example, random guessing will give you exactly 0 points, at least in expectation.

Algorithmic Problems

Camelot

King Arthur expects n knights for an annual dinner at Camelot, including himself. All the knights are to be seated around the same, huge Round Table.

Knights prefer to talk down to people, so if the party is going to be a success, each knight must be seated next to at least one younger knight. (Otherwise, if a knight is seated between two older knights, he will get bored and start singing, ruining the evening for everybody.) Arthur is an exception: everybody would be pleased to sit next to Arthur, no matter their age.

Input

The input consists of a line containing n, followed by of a list of $n-1$ names and birthdays, one on each line. No knights are born on the same day.

Output

A seating order around the table, starting with Arthur.

Example 1
Input

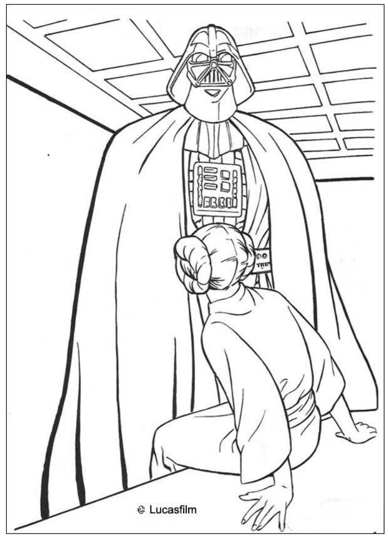
```
150
Gawain, 1 June 874
Lancelot, 12 September 823
... (146 more names)
Parceval, 1 December 890
```

Output

```
Arthur
Parceval
... (147 more names)
Robert
```

Torture

You are an evil overlord who wants to get princess Orga Leiana to reveal her secrets. At your disposal is a huge weapon that can destroy entire stars. To demonstrate your powers you will destroy a star in front of the princess's eyes and then quickly fly the weapon to a close-by star and threaten to blow that one up, too.



You want the interrogation to be over quickly, so you want to be able to move the weapon to the second star very quickly. (It doesn't matter how long it takes to get to the first star.)

Input

The input starts with a number n between 2 and 100000, giving the number of stars in the Evil Galactic Empire. On each following line, two numbers x and y give the co-ordinates of each star. (The universe is two-dimensional[1]. The distance between two stars is the Euclidian distance.)

Output

Output the text "Blow up A then fly to B" where A and B are indices of stars $1 \ldots n$, and the trip from A to B is as fast as possible. (If more than one optimal solution exists, any one will do.)

Example

Input

```
3
5 5
6 5
100 100
```

Output

```
Blow up 1 then fly to 2
```

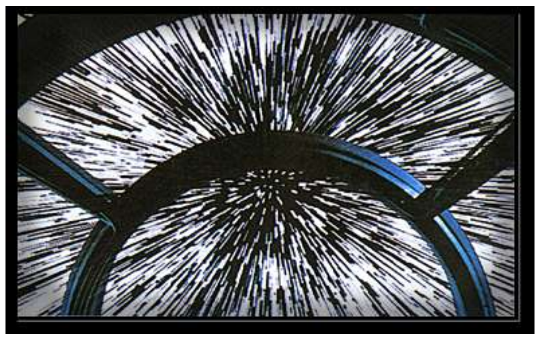
[1] So are all the characters in the plot

Hyperspace

You are an evil galactic overlord who wants to destroy your enemy's empire. At your disposal is a huge weapon that can destroy entire stars. The physics of interstellar navigation are complicated, but boil down to two rules:

1. you cannot fly into empty space, only from star to star
2. certain pairs of stars are blocked by various dangers in hyperspace, meaning you can't fly directly from one to the other. (You have a map of these things.)

Once you arrive at a star, the weapon is programmed to automatically destroy the star. For simplicity, you can assume that the first trip (from your own home to a star in the enemy's empire) is not blocked by any hyperspace problems, and you don't care about flying the weapon home when the job is done.



Input

The input starts with a number n between 2 and 1000, giving the number of stars you want to blow up (the stars are called $1, 2, \ldots, n$ for simplicity), and a number m between 0 and $n(n-1)/2$, giving the number of hyperspace dangers. Then follows, line by line, a list of all the dangers in hyperspace in the form "danger: x y", where x and y are star names and danger is a description of the problem.

Output

Output a flight plan of n integers, or the text "can't be done."

Example 1

Input

```
5 4
Hyperspace storm: 1 3
Hyperspace storm: 1 4
Borg: 3 5
Space-ship eating dragon: 2 5
```

Output

```
1 2 3 4 5
```

Example 2

Input

```
5 4
Hyperspace storm: 1 3
Acid cloud: 1 4
Tribbles: 1 2
Space-ship eating dragon: 1 5
```

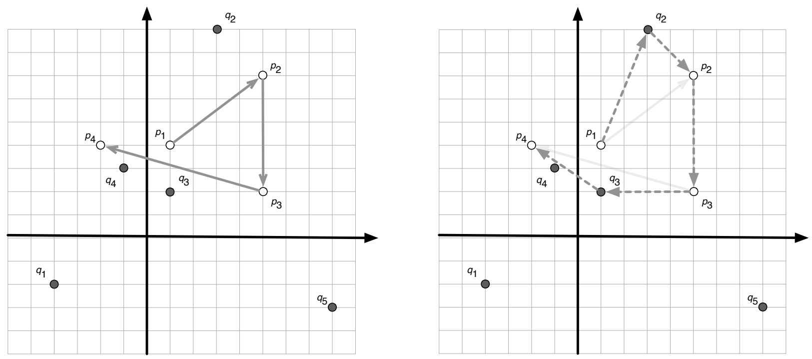
Output

```
Can't be done
```

Dog

Hunter Bob often walks with his dog Ralph.

Bob walks with constant speed and his route is a polygonal line (possibly self-intersecting) $p_1, \ldots, p_n$. See the figure to the left:



Ralph's route is more flexible. However, he always meets his master at each of the the specified n points. In particular, Ralph starts simultaneously with Bob at p_1 and finishes simultaneously with Bob at p_n.

Ralph can travel at a speed that is up to two times greater than Bob's. Between meetings with Bob, Ralph cheerfully inspects trees, whose location is given by m points $q_1, \ldots, q_m$. Ralph wants to maximise the number of trees he gets to inspect, but he only visits at most one tree between every meeting with Bob, and never inspects the same tree twice. In the above figure to the right, an optimal route for Bob is $p_1, q_2, p_2, p_3, q_3, p_4$, allowing him to visit 2 trees.

Input

The first line contains two integers n and m, separated by a space. The second line contains n pairs of integers, separated by spaces, that represent the x- and y-coordinates of $p_1, \ldots, p_n$. The third line contains m pairs of integers, separated by spaces, representing the trees.

All points in the input file are different and their coordinates are integers not greater than 1000 by the absolute value.

Output

The first line of each dataset should contain the single integer k – the number of vertices of the best dog's route. The second line should contain k pairs of coordinates, separated by spaces, that represent this route. If there are several such routes, then you may write any of them.

Example 1

Input

```
4 5
1 4 5 7 5 2 -2 4
-4 -2 3 9 1 2 -1 3 8 -3
```

Output

```
6
1 4 3 9 5 7 5 2 1 2 -2 4
```

ABBA

Your task is to transform a given string into a palindrome. (A palindrome is a string that is the same left-to-right as right-to-left, such as “ABBA”,“UBU”, “GARNXNRAG”, “ABLEWASIEREISAWELBA”, and even “XX” and “L”.[2])

The operations you’re allowed to use are (1) remove any letter and (2) replace any letter by another letter. You need to complete the task in as few operations as possible. For example, you can transform “GNGN” into a palindrome by using two operations: “GNGN” → “GNNN” → “GNNG”, but you could also do it using just one operation: “GNGN” → “GNG”.

Input

The input contains a strings of up to 1000 characters.

Output

Write the smallest number of operations sufficient to transform the string into a palindrome.

Input `tanbirahmed`

Output `5`

[2]For the formalists: yes, the empty string is a palindrome.

Exam Questions

Analysis of algorithms

1. Let $f(n) = (n^2 + 2n)\log n + \frac{1}{1000} n^{5/2}$. True of false?

(a) (*1 pt.*) $f(n) = O(n^3)$

$\boxed{\text{A}}$ true $\boxed{\text{B}}$ false

(b) (*1 pt.*) $f(n) = O(n^{5/2})$

$\boxed{\text{A}}$ true $\boxed{\text{B}}$ false

(c) (*1 pt.*) $f(n) = O(n^2 \log n)$

$\boxed{\text{A}}$ true $\boxed{\text{B}}$ false

2. Consider the following piece of code:

```
1: for i = 1 to n
2:     for j = 1 to n
3:         print i + j;
```

(a) (*2 pt.*) What is the running time? (Choose the smallest correct estimate.) Assume **print** takes constant time.

$\boxed{\text{A}}$ $O(\log n)$ $\boxed{\text{B}}$ $O(\sqrt{n}\log n)$ $\boxed{\text{C}}$ $O(n)$ $\boxed{\text{D}}$ $O(n\log n)$ $\boxed{\text{E}}$ $O(n^{3/2})$
$\boxed{\text{F}}$ $O(n^2)$ $\boxed{\text{G}}$ $O(n^2\log n)$ $\boxed{\text{H}}$ $O(n^3)$ $\boxed{\text{I}}$ $O(n!)$ $\boxed{\text{J}}$ $O(2^n\log n)$

(b) (*1 pt.*) Assume I changed line 4 to "**for** $j = 1$ **to** i". Then the running time for the whole algorithm is asymptotically

$\boxed{\text{A}}$ faster $\boxed{\text{B}}$ slower $\boxed{\text{C}}$ the same

3. Assume you have a data structure that maintains a set S under insertion and deletion. The operation "**insert**(s, S)" makes sure that $s \in S$ holds, and the operation "**remove**(S)" chooses some $s \in S$ (assuming S is not empty) and removes it.

Consider the following piece of code, starting with an empty set S.

```
1: for i = 1 to n
2:     insert (i, S)
3: for i = 1 to n
4:     remove (S)
```

(a) (*1 pt.*) Assume **insert** (i, S) takes time $O(|S|)$, and **remove**(S) takes constant time. Then the running time is: (Choose the smallest correct estimate.)

$\boxed{\text{A}}$ $O(\log n)$ $\boxed{\text{B}}$ $O(\sqrt{n}\log n)$ $\boxed{\text{C}}$ $O(n)$ $\boxed{\text{D}}$ $O(n\log n)$ $\boxed{\text{E}}$ $O(n^{3/2})$
$\boxed{\text{F}}$ $O(n^2)$ $\boxed{\text{G}}$ $O(n^2\log n)$ $\boxed{\text{H}}$ $O(n^3)$ $\boxed{\text{I}}$ $O(n!)$ $\boxed{\text{J}}$ $O(2^n\log n)$

(b) (*1 pt.*) Assume **insert** (i, S) and **remove**(S) both take time $O(|S|)$. Then the running time is: (Choose the smallest correct estimate.)

$\boxed{\text{A}}$ $O(\log n)$ $\boxed{\text{B}}$ $O(\sqrt{n}\log n)$ $\boxed{\text{C}}$ $O(n)$ $\boxed{\text{D}}$ $O(n\log n)$ $\boxed{\text{E}}$ $O(n^{3/2})$
$\boxed{\text{F}}$ $O(n^2)$ $\boxed{\text{G}}$ $O(n^2\log n)$ $\boxed{\text{H}}$ $O(n^3)$ $\boxed{\text{I}}$ $O(n!)$ $\boxed{\text{J}}$ $O(2^n\log n)$

(c) (*1 pt.*) Assume **insert** (i, S) takes constant time, and **remove**(S) takes time $O(\log |S|)$. Then the running time is: (Choose the smallest correct estimate.)

A $O(\log n)$ B $O(\sqrt{n} \log n)$ C $O(n)$ D $O(n \log n)$ E $O(n^{3/2})$
F $O(n^2)$ G $O(n^2 \log n)$ H $O(n^3)$ I $O(n!)$ J $O(2^n \log n)$

4. Consider the following piece of code:

```
int f(int n) {
    j = 0
    for i = 0 to n { j = j + i; }
    if j > 9 then { return f(n − 1) + n/2; }
    else return 3;
}
```

(a) (*3 pt.*) Which recurrence relation best characterises the running time of this method?

A $T(n) = T(n-1) + O(1)$ B $T(n) = 2T(n/2) + O(\log n)$
C $T(n) = T(n-1) + T(n/2)$ D $T(n) = T(n-1) + O(n)$
E $T(n) = 2T(n-1) + O(1)$ F $T(n) = 2T(n/2) + O(\log n)$

(b) (*1 pt.*) Find the solution to $T(n) = \frac{1}{2} + T(n-1), \quad T(0) = 0$. Give the smallest correct answer.

A $O(\sqrt{n} \log n)$ B $O(n)$ C $O(n \log n)$ D $O(n^{3/2})$ E $O(n^2)$
F $O(n^2 \log n)$ G $O(n!)$ H $O(2^n \log n)$ I $O(2^{2^n})$

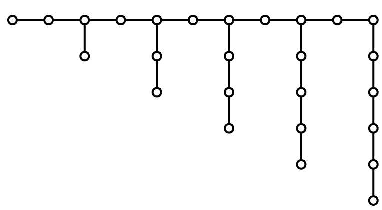
(c) (*1 pt.*) Find the solution to $T(n) = \big(T(n-1)\big)^2, \quad T(1) = 2$. Give the smallest correct answer.

A $O(\sqrt{n} \log n)$ B $O(n)$ C $O(n \log n)$ D $O(n^{3/2})$ E $O(n^2)$
F $O(n^2 \log n)$ G $O(n!)$ H $O(2^n \log n)$ I $O(2^{2^n})$

Graphs

5. For integer $r \geq 1$, the rth *comb graph* $C_r = (V_r, E_r)$ consists of r (vertical) paths of $1, \ldots, r$ vertices connected to a single (horizontal) path of $2r - 1$ vertices like in this example for $r = 6$:

$C_6 =$



(a) (*1 pt.*) The number of vertices $|V_r|$ in C_r is

A $\binom{r}{2} + 2r - 1$ B $r^2 - 10$ C $5r - 4$

(b) (*1 pt.*) The number of edges $|E_r|$ in C_r is

A $|V_r| - 1$ B $|V_r| + r - 5$ C $|V_r| - \frac{1}{6}r$

(c) (*1 pt.*) C_r is a tree

A True B False

(d) (*1 pt.*) Each vertex in C_r has degree

A $\Theta(\log r)$ B $O(1)$ C $r/2$ D $\Theta(r)$ E 7

(e) (*1 pt.*) A maximum independent set in C_r has size

A $\Theta(\log r)$ B $O(1)$ C r D $\Theta(r^2)$ E 7

Divide-and-conquer

6.

(a) (*3 pt.*) The following problem can be solved by divide-and-conquer (but not by a greedy algorithm).

[A] Camelot [B] Torture [C] Hyperspace [D] Dog [E] ABBA

(b) (*1 pt.*) For that, split the n input objects into two equal sized sets according to their

[A] x-coordinate [B] age [C] Euclidian distance from the first object [D] number of incident dangers

[E] alphanumeric value [F] index

and solve the problem recursively.

(c) (*1 pt.*) The recursion depth is (only give the smallest correct answer)

[A] $O(\log n)$ [B] $O(\log\log n)$ [C] $O(\log^2 n)$ [D] $O(1)$

[E] $O(n)$

(d) (*1 pt.*) To make the "conquer"-step work in less than quadratic time one can sort the two halves individually by

[A] y-coordinate [B] age [C] Euclidian distance from the first object [D] number of incident dangers

[E] alphanumeric value [F] index

and perform a constant amount of additional work.[3]

(e) (*2 pt.*) This way, the reccurence relation that best characterises the whole algorithm is

[A] $T(n) = 2T(n/2) + O(1)$ [B] $T(n) = 2T(n/2) + O(\log n)$

[C] $T(n) = 2T(n/2) + O(\sqrt{n})$ [D] $T(n) = 2T(n/2) + O(n\log n)$

[E] $T(n) = 2T(n/2) + O(n\log^2 n)$ [F] $T(n) = 2T(n/2) + O(n^2)$

(f) (*2 pt.*) The resulting running time for the whole algorithm is

[A] $O(n\log\log n)$

[B] $O(n\log^2 n)$

[C] $O(n)$

[D] $O(n^2)$

Greedy

7.

(a) (*3 pt.*) The following problem admits a greedy algorithm:

[A] Camelot [B] Torture [C] Hyperspace [D] Dog [E] ABBA

(b) (*2 pt.*) The objects are processed in order of their

[A] x-coordinate [B] age [C] Euclidian distance from the first object [D] number of incident dangers

[E] row number [F] length [G] index

(c) (*1 pt.*) The running time is (give the smallest possible)

[A] $O(n)$. [B] $O(n\log n)$.

[3] Maybe you know a trick to make it even faster. Ignore that, just stick to the current exercise.

Dynamic programming

8.

(a) (*3 pt.*) One of the problems is solved by dynamic programming. (But not by a faster approach such as greedy or divide-and-conquer.)

A Camelot   B Torture   C Hyperspace   D Dog   E ABBA

(b) (*3 pt.*) Following the book's notation, we let $\mathrm{OPT}(i,j)$ denote the value of a partial solution. Then OPT satisfies[4]

A

$$\mathrm{OPT}(i,j) = \min\{\mathrm{OPT}(i-1,j), \mathrm{OPT}(i-1,j-1) + f(i,j)\}$$

B

$$\mathrm{OPT}(i,j) = \begin{cases} \mathrm{OPT}(i+1,j-1)\,, & \text{if } f(i,j), \\ 1+\min\big(\mathrm{OPT}(i+1,j), \mathrm{OPT}(i,j-1), \mathrm{OPT}(i+1,j-1)\big)\,, & \text{otherwise.} \end{cases}$$

C

$$\mathrm{OPT}(i,j) = \min_{j\le k\le N}\{\,\mathrm{OPT}(i-1,k) + f(i,k)\,\}$$

D

$$\mathrm{OPT}(i,j) = \max\{\,\mathrm{OPT}(f(i,j),j), \mathrm{OPT}(i-1,j)\,\}$$

E

$$\mathrm{OPT}(i,j) = \min_{j-1\le k\le j+1}\{\,\mathrm{OPT}(i-1,k) + f(i-1,k)\,\}$$

F

$$\mathrm{OPT}(i,j) = \begin{cases} \mathrm{OPT}(i-1,j) & \text{if } f(i,j) \\ \max\{\mathrm{OPT}(i-1,j), 1+\mathrm{OPT}(i-1,j-1)\} & \text{otherwise} \end{cases}$$

(c) (*2 pt.*) where $f(i,j)$ is

A height difference between knights i and j

B the distance between stars i and j

C $i > j$

D the number of hyperspace dangers between stars i and j

E $\frac{1}{2}(|p_i - q_j| + |q_j - p_{i+1}|)$

F true iff and only if the ith letter equals the jth

(d) (*1 pt.*) The resulting running time is

A $O(n)$ B $O(n\log n)$ C $O(n^2)$ D $O(n^2\log n)$
E $O(m)$ F $O(n\log m)$ G $O(nm)$ H $O(nm^2)$

Network flow

[4] There may be other cases, in particular, boundary conditions such as maybe for $\mathrm{OPT}(1,1)$ or $\mathrm{OPT}(N,0)$ or $i=j$, etc. Don't worry about them. This question is just about the most central part of the recurrence relation, otherwise this exercise becomes too large.

9.

(a) (*3 pt.*) One of the problems in the set is easily solved by a reduction to network flow. That problem is

A Camelot B Torture C Hyperspace D Dog E ABBA

(b) (*2 pt.*) The network consists of a node for[5]

A each knight B each knight and each table position
C each star D each star and each hyperspace danger
E each point p_i and each tree q_i F each letter

(c) (*1 pt.*) The total number of nodes (including the soure and the sink), in terms of the parameters of the original problem, is

A $m+n+2$ B $m+n$ C 2 D m
E n F $O(1)$ G mn H $mn \log n$

(d) (*3 pt.*) Describe the reduction on the back of this page, or on a separate piece of paper. Be ridiculously precise about how the nodes are connected and directed, and what the capacities are. Do this in general (use words like "every node corresponding to a giraffe is connected to every node corresponding to a letter by an undirected arc of capacity the length of the neck"), and also draw a small, but *complete* example for an example instance. What does a maximum flow mean in terms of the original problem, and what size does it have in terms of the original parameters?

Computational complexity

These questions are about *Dog*, and include questions about NP. Don't take this as an indication that Dog may or may not be NP-hard.

10. *(Decision version.)* The input to the decision version of Dog includes

(a) (*1 pt.*) the points $p_1, \ldots, p_n$ and $q_1, \ldots, q_m$,

A true B false

(b) (*1 pt.*) an optimal route for Ralph

A true B false

(c) (*1 pt.*) an integer k

A true B false

(d) (*1 pt.*) The output to the decision version of Gophers is

A "yes," if there is way for Ralph to visit k trees

B "yes," if it is possible for Ralph to visit all m trees

C a single integer, which is the maximum number of trees that Ralph can visit

D an optimal route for Ralph

E the points $p_1, \ldots, p_n$ and $q_1, \ldots, q_m$

[5] Apart from these, there may be a *constant* number of extra nodes, such as a source and a sink, or a single node representing the table, or the superweapon, or the two banks of the river, etc.

11. *Membership in NP.* The decision version of Dog is easily seen to be in NP, because it admits a certificate.

(a) (*2 pt.*) The certificate includes, (apart from the input already listed in answer above,) the following information

- [A] "yes," if there is way for Ralph to visit k trees
- [B] "yes," if it is possible for Ralph to visit all m trees
- [C] a single integer, which is the maximum number of trees that Ralph can visit
- [D] an optimal route for Ralph
- [E] the points $p_1, \ldots, p_n$ and $q_1, \ldots, q_m$

(b) (*2 pt.*) The certificate has length (choose the smallest possible)

[A] $O(n+m)$ [B] $O(n^2)$ [C] $O(nm)$ [D] $O(m^2)$

NP-hardness

One of the problems in the set is NP-hard.[6]

12.

(a) (*3 pt.*) The following problem (called P_1) is NP-hard:

[A] Camelot [B] Torture [C] Hyperspace [D] Dog [E] ABBA

(b) (*3 pt.*) The easiest way to see that is to take the following NP-hard problem, called P_2,

[A] Graph colouring [B] 3-dim. matching [C] Independent set [D] Set packing
[E] Vertex cover [F] 3-satisfiability [G] Travelling salesman [H] Hamiltonian cycle

(c) (*2 pt.*) and prove

[A] $P_1 \leq_P P_2$ [B] $P_2 \leq_P P_1$

(d) (*1 pt.*) For this, an arbitrary instance of

[A] Camelot [B] Torture [C] Hyperspace [D] Dog
[E] ABBA [F] Graph colouring [G] 3-dim. matching [H] Independent set
[I] Set packing [J] Vertex cover [K] 3-satisfiability [L] Travelling salesman
[M] Hamiltonian cycle

(e) (*1 pt.*) is transformed into an instance of

[A] Camelot [B] Torture [C] Hyperspace [D] Dog
[E] ABBA [F] Graph colouring [G] 3-dim. matching [H] Independent set
[I] Set packing [J] Vertex cover [K] 3-satisfiability [L] Travelling salesman
[M] Hamiltonian cycle

(f) (*4 pt.*) Describe the reduction on the back of this page, or on a separate piece of paper. Do this both in general and for a small but complete example. In particular, be ridiculously precise about the paramters of the instance you produce (for example number of vertices, edges, sets, colors) in terms of the parameters of the original instance, what the solution of the transformed instance means in terms of the original instance, etc.

---

[6]If P = NP then *all* these problems are NP-hard. Thus, in order to avoid unproductive (but hypothetically correct) answers from smart alecks, this section assumes that P $\neq$ NP.