CS 91 USACO

Bronze Division

Unit 1: Introduction to Competitive Programming

LECTURE 2: AD HOC ALGORITHMS

DR. ERIC CHOU

IEEE SENIOR MEMBER

Objectives

- What is ad hoc problems?
- Transaction-based problem. (Amortized Analysis)
- Practice: Greedy Gift Givers
- City Block Distance and Fining Minimum
- Exam: December 14, P1

What is Ad Hoc Algorithms?

SECTION 1

Programming Contest Problem Types

- Hal Burch conducted an analysis over spring break of 1999 and made an amazing discovery: there are only 16 types of programming contest problems! Furthermore, the top several comprise almost 80% of the problems seen at the IOI.

Programming Contest Problem Types

- Dynamic Programming
- Greedy
- Complete Search
- Flood Fill
- Shortest Path
- Recursive Search Techniques
- Minimum Spanning Tree
- Knapsack
- Computational Geometry
- Network Flow
- Eulerian Path
- Two-Dimensional Convex Hull
- BigNums
- Heuristic Search
- Approximate Search
- Ad Hoc Problems

Programming Contest Problem Types

- The most challenging problems are Combination Problems which involve a loop (combinations, subsets, etc.) around one of the above algorithms - or even a loop of one algorithm with another inside it. These seem extraordinarily tricky to get right, even though conceptually they are ``obvious''.
- If you can master solving just 40% of these problem types, you can almost guarantee a silver medal at the IOI. Mastering 80% moves you into the gold range almost for sure. Of course, `mastery' is a tough nut to crack! We'll be supplying a plethora of problems so that you can hone your skills in the quest for international fame.

Ad Hoc Problems

- `Ad hoc' problems are those whose algorithms do not fall into standard categories with well-studied solutions. Each ad hoc problem is different; no specific or general techniques exist to solve them.
- Of course, this makes the problems the `fun' ones, since each one presents a new challenge. The solutions might require a novel data structure or an unusual set of loops or conditionals. Sometimes they require special combinations that are rare or at least rarely encountered.

Ad Hoc Problems

- Ad hoc problems usually require careful reading and usually yield to an attack that revolves around carefully sequencing the instructions given in the problem.
- Ad hoc problems can still require reasonable optimizations and at least a degree of analysis that enables one to avoid loops nested five deep, for example.
- More ad hoc problems appear on this web site than any other kind of problem. Always be ready for an ad hoc problem if you can not classify a problem as one of the other standard types (to be listed later).

Practice: Greedy Gift Givers (gift1)

SECTION 2

Problem Statement

- A group of NP (2 ≤ NP ≤ 10) uniquely named friends has decided to exchange gifts of money. Each of these friends might or might not give some money to some or all of the other friends (although some might be cheap and give to no one). Likewise, each friend might or might not receive money from any or all of the other friends. Your goal is to deduce how much more money each person receives than they give.

Problem Statement

- The rules for gift-giving are potentially different than you might expect. Each person goes to the bank (or any other source of money) to get a certain amount of money to give and divides this money evenly among all those to whom he or she is giving a gift. No fractional money is available, so dividing 7 among 2 friends would be 3 each for the friends with 1 left over – that 1 left over goes into the giver's "account". All the participants' gift accounts start at 0 and are decreased by money given and increased by money received.

Problem Statement

- In any group of friends, some people are more giving than others (or at least may have more acquaintances) and some people have more money than others.
- Given:
  - a group of friends, no one of whom has a name longer than 14 characters,
  - the money each person in the group spends on gifts, and
  - a (sub)list of friends to whom each person gives gifts,
  - determine how much money each person ends up with.

INPUT FORMAT (file gift1.in):

- **Line 1** A single integer, NP
- **Line 2..NP+1** Line i+1 contains the name of group member i
- **Line NP+2..end** NP groups of lines organized like this:
- The first line of each group tells the person's name who will be giving gifts.
- The second line in the group contains two numbers:
- The amount of money (in the range 0..2000) to be divided into gifts by the giver
- NG_i ($0 \le NG_i \le NP$), the number of people to whom the giver will give gifts
- If NGi is nonzero, each of the next NG_i lines lists the name of a recipient of a gift; recipients are not repeated in a single giver's list.

INPUT FORMAT (file gift1.in):

SAMPLE INPUT:

```
5
dave
laura
owen
vick
amr
dave
200 3
laura
owen
vick
owen
500 1
dave
amr
150 2
vick
owen
laura
0 2
amr
vick
vick
0 0
```

OUTPUT FORMAT (gift1.out):

- The output is NP lines, each with the name of a person followed by a single blank followed by the net gain or loss (final_money_value - initial_money_value) for that person. The names should be printed in the same order they appear starting on line 2 of the input.
- All gifts are integers. Each person gives the same integer amount of money to each friend to whom any money is given, and gives as much as possible that meets this constraint. Any money not given is kept by the giver.

SAMPLE OUTPUT:

```
dave 302
laura 66
owen -359
vick 141
amr -150
```

Nature of the Problem

- Basic Text Processing
- Simple Amortized Analysis

December 2014 Bronze problem 1: Marathon

SECTION 3

Problem Statement

- Unhappy with the poor health of his cows, Farmer John enrolls them in an assortment of different physical fitness activities. His prize cow Bessie is enrolled in a running class, where she is eventually expected to run a marathon through the downtown area of the city near Farmer John's farm!
- The marathon course consists of N checkpoints (3 <= N <= 100,000) to be visited in sequence, where checkpoint 1 is the starting location and checkpoint N is the finish. Bessie is supposed to visit all of these checkpoints one by one, but being the lazy cow she is, she decides that she will skip up to one checkpoint in order to shorten her total journey. She cannot skip checkpoints 1 or N, however, since that would be too noticeable.

Problem Statement

- Please help Bessie find the minimum distance that she has to run if she can skip up to one checkpoint.
- Note that since the course is set in a downtown area with a grid of streets, the distance between two checkpoints at locations (x1, y1) and (x2, y2) is given by |x1-x2| + |y1-y2|. This way of measuring distance -- by the difference in x plus the difference in y – is sometimes known as "Manhattan" distance because it reflects the fact that in a downtown grid, you can travel parallel to the x or y axes, but you cannot travel along a direct line "as the crow flies".

INPUT FORMAT (file marathon.in):

- The first line gives the value of N.
- The next N lines each contain two space-separated integers, x and y, representing a checkpoint (-1000 <= x <= 1000, -1000 <= y <= 1000). The checkpoints are given in the order that they must be visited. Note that the course might cross over itself several times, with several checkpoints occurring at the same physical location. When Bessie skips such a checkpoint, she only skips one instance of the checkpoint -- she does not skip every checkpoint occurring at the same location.

SAMPLE INPUT:

```
4
0 0
8 3
11 -1
10 0
```

OUTPUT FORMAT (cowtip.out):

- Output the minimum distance that Bessie can run by skipping up to one checkpoint. Don't forget to end your output with a newline. In the sample case shown here, skipping the checkpoint at (8, 3) leads to the minimum total distance of 14.

SAMPLE OUTPUT:

```
14
```

Nature of Problem

- Finding sum and minimum value.

```
0  0
8  3
11  -1
10  0
```