Петрозаводский государственный университет
Математический факультет
Кафедра информатики и математического обеспечения

Курсовая работа

Идентификация маршрута мобильного пользователя по набору GPS координат.

Выполнил: студент 2 курса
Сало А. Ю.

Научный руководитель: кандидат технических наук, доцент кафедры ИМО
Богоявленская О. Ю.

Представлена: ______________
2002 г.

Оценка оформления и сроков представления курсовой работы:

Оценка публичной защиты работы:

Оценка промежуточного отчета:

Итоговая оценка:

Отв. преподаватель:

Петрозаводск
2002

Содержание

1 Введение

Идентификация маршрута мобильного пользователя по набору его GPS координат — задача, тесно связанная с задачей упрощения линии в геометрии и картографии. Идентификация предшествует прогнозированию характеристик производительности при мобильной передаче данных в коммуникационных системах. Эта задача актуальна, и целью работы является получение достаточно эффективного алгоритма для ее решения.

2 Постановка проблемы

Идентификация маршрута мобильного пользователя происходит с использованием координат, предоставляемых GPS (Global Positioning System), или просто GPS координат. Во время движения пользователя через некоторые промежутки времени происходит вычисление его текущих GPS координат. Таким образом, мы получаем последовательность точек в пространстве, которые в какой-то степени отображают перемещение пользователя. Задача заключается в том, чтобы с заданной точностью представить маршрут пользователя в виде ломаной. Эта задача подразделяется на две части. Во-первых, необходимо получить набор точек с использованием GPS. При этом точки должны находиться в удобной для дальнейших вычислений системе координат. Во-вторых, необходимо вычислить и оставить точки, являющиеся точками поворота в маршруте пользователя, который надо с некоторой точностью представить в виде ломаной, а остальные точки отбросить.

2.1 Использование GPS

GPS предоставляет данные о положении пользователя в двух стандартных системах координат: в декартовой трехмерной системе координат, зафиксированной относительно Земли[1], и в системе географических координат (широта, долгота и высота над поверхностью Земли[2]. Существуют формулы для перевода ECEF XYZ координат в географические и обратно. Однако для нас было бы удобнее использовать координаты точки в двумерной декартовой системе координат (с учетом того, что идентифицировать маршрут пользователя предполагается на ограниченной территории, т. е. поверхность Земли можно рассматривать как плоскость, и, кроме того, предполагается, что перепадами высот в данной местности можно пренебречь). В принципе, можно использовать непосредственно ECEF XYZ, т. е. трехмерные декартовы координаты.

[1] Cartesian Earth-Centered, EarthFixed (ECEF) X, Y, Z

[2] Geodetic Latitude, Longtitude and Height

При этом усложнится процедура определения расстояния между точками, станет труднее определять точки поворота. С другой стороны, не потребуется дополнительного времени на перевод из одной системы координат в другую, а главное, можно будет избежать грубых ошибок, связанных с такими переводами. Возможно также использование географических координат без учета высоты над поверхностью Земли.

Важным является то, что GPS приемники могут предоставлять координаты положения мобильного пользователя в иной, требуемой самим пользователем системе координат. Остается определить правила построения системы координат для данного мобильного пользователя на данной локальной территории. Это является темой для будующих исследований.

2.2 Алгоритм Дугласа—Пекера

Вторая часть проблемы идентификации маршрута мобильного пользователя по его GPS координатам тесно связана с проблемой приближения или упрощения линии[3] в картографии. Наиболее распространенным для решения этой задачи является алгоритм Дугласа-Пекера.

Базовый алгоритм Дугласа-Пекера довольно прост и легко реализуем с использованием рекурсии. Идея его заключается в следующем: пусть необходимо приблизить отрезок $\overline{V_i V_j}$ с точностью ϵ. Если расстояние самой удаленной от этого отрезка вершины не превосходит ϵ, тогда он упрощен. В противном случае, берем эту вершину в качестве разделяющей и рекурсивно приближаем два получившихся отрезка.

Дан массив вершин V, вызов DPbasic (V, i, j) приближает часть ломаной от V_i до V_j.

Procedure DPbasic (V, i, j)

1. Найти вершину V_f, максимально удаленную от $\overline{V_i V_j}$. Присвоить $dist$ значение расстояния до этой точки.
2. if $dist > \epsilon$ then
   - 3a. DPbasic (V, i, f)
   - 3b. DPbasic (V, f, j)

   else
   - 4. Output($\overline{V_i V_j}$)

Базовый алгоритм приближения линии.

Оценка сложности этого алгоритма дает $O(n^2)$ для количества точек n. Это, конечно, неудовлетворительно, т. к. при большом количестве

[3] line simplification problem

точек время работы существенно возрастает. Есть несколько другой алгоритм, основное различие которого с алгоритмом Дугласа-Пекера состоит в ином способе нахождения точки разделения, т. е. точки поворота. Он основывается не на непосредственном последовательном вычислении расстояний до каждой точки с последующим определением наиболее удаленной, а на утверждении, что максимально удаленная точка находится на выпуклом многоугольнике, который содержит все точки ломаной. Сначала определяются точки, которые являются вершинами этого многоугольника, затем из них выбирается та, которая является максимально удаленной. Навание алгоритма — path hulls based algorithm — можно перевести как алгоритм, основанный на построении сопутствующих многоугольников. Время работы этого алгоритма в худшем случае пропорционально $O(n \log_2 n)$, что равняется времени работы базового алгоритма Дугласа-Пекера в лучшем случае. Такое время работы является удовлетворительным, но требуется дополнительная память для использования специальной структуры данных.

Найти точку поворота V_f, наиболее удаленную от линии $\overline{V_i V_j}$.
Вернуть расстояние $dist$ вершины V_f.

Procedure FindSplit (V, i, j)
    set $dist = 0$
    for $k = i + 1$ to $k = j - 1$ do

$$dist\,Vk = \left(\frac{\begin{vmatrix} 1 & V_i \cdot x & V_i \cdot y \\ 1 & V_j \cdot x & V_j \cdot y \\ 1 & V_k \cdot x & V_k \cdot y \end{vmatrix}}{\sqrt{(V_i \cdot x - V_j \cdot x)^2 + (V_i \cdot y - V_j \cdot y)^2}} \right)$$

        if $dist\,Vk \geq dist$ then
            set $dist = dist\,Vk$
            set $f = k$

Нахождение точки поворота.

Однако, на мой взгляд, алгоритм Дугласа-Пекера, равно как и усовершенствованный алгоритм, обладает недостатками, которые делают его непосредственное применение неподходящим для идентификации маршрута мобильного пользователя.

Во-первых, самым главным недостатком алгоритма яляется то, что он применим только для ломаных, не имеющих самопересечений. Авторы случаи с самопересечениями просто не рассматривают, только оговаривают, что верный результат гарантирован лишь при их отсутствии. Очевидно, что на практике такие случаи не только могут, но и будут появляться, поэтому алгоритм будет приводить к неверным результатам.

Во-вторых, приходится при добавлении каждой новой точки пересчитывать заново все точки поворота. Несмотря на то, что сложность усовершенствованного алгоритма $O(n \log_2 n)$, затраты по времени могут оказаться значительными. Кроме того, приходится по многу раз пересчитывать уже найденные точки поворота, т. е. выполнять по нескольку раз уже выполненную до этого работу.

Выход из этой ситуации достаточно прост: применяем алгоритм каждый раз с получением новой координаты до тех пор, пока не найдем новую точку поворота; запоминаем эту точку; снова применяем алгоритм, но теперь не рассматриваем те точки, которые отсекаются запомненной точкой; находим следующую точку поворота, запоминаем ее и т. д. Это решение будет не всегда оптимальным, т. к. некоторые полученные точки не будут являться точками поворота, если применить алгоритм Дугласа—Пекера непосредственно, но зато достигается существенное уменьшение времени работы программы. Следовательно, для упрощения линии становится возможным использовать базовый алгоритм Дугласа—Пекера, который проще реализовать, т. е. не нужно дополнительно описывать структуры данных, используемые в усовершенствованном алгоритме, и работающие с этими структурами функции. Нахождение точки поворота сводится к нахождению максимального из расстояний от каждой точки до отрезка. При этом для малого количества точек время его работы не намного отличается от времени работы усовершенствованного алгоритма, и он не требует дополнительной памяти, как усовершенствованный алгоритм. Такое решение является наиболее выгодным и с точки зрения скорости выполнения, и с точки зрения экономии памяти.

3 Заключение

В работе рассматривалась задача идентификации маршрута мобильного пользователя по набору его GPS координат. Для ее решения применялся алгоритм Дугласа—Пекера упрощения линии. Алгоритм был модифицирован с учетом того, что координаты мобильного пользователя считываются последовательно. Модифицированный алгоритм реализован на языке C.

4 Код программы на C

```
/*****************************************************************
*
*   Douglas—Peucker Line Simplification Algorithm.
*
```

```
*   Author: Andrew Salo <salo@cs.karelia.ru>
*       Petrozavodsk State University
*       Department of Computer Science
*
*   Date: 2002
*
*************************************************************
```

Программа является реализацией алгоритма Дугласа—Пекера для упрощения линии с учетом того, что упрощение линии происходит динамически, т. е. точки не даны все сразу, а считываются одна за другой. Найденные точки поворота запоминаются и больше не рассматриваются при последующих упрощениях линии.

Компиляцию следует выполнять с ключом `-lm`, т. к. программа вызывает функцию sqrt.

Процедуры:

`int dpbase (int i, int j)` — нахождение наиболее удаленной точки части ломаной от i до j включительно.

`void readfile (FILE *f)` — чтение координат очередной точки из файла, вызов функцию `dpbase`.

`void writefile (FILE *f)` — запись в файл точек, являющихся вершинами упрощенной ломаной.

`int main (int argc, char **argv)` — обработка аргументов командной строки, открытие и закрытие файлов, вызов `readfile` и `writefile`, инициализация указателей и обработка ошибок.

```
*************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <errno.h>

#define MAXPOINTS 10000

struct dppoint {        /* Точка на плоскости. */
   float x;
   float y;
} pts [MAXPOINTS], simp [MAXPOINTS];

int n, simpptr;

float e;      /* Точность приближения линии. */
```

```
/* Функция dpbase — нахождение наиболее удаленной точки.
Аргументы: i — начальная точка приближаемой ломаной, j — конечная
точка.
Возвращает: номер точки, находящейся на максимальном расстоянии от
ij, если это расстояние превосходит заданную точность, иначе — 0.*/
int dpbase (int i, int j)
{
   /* Максимальное и текущее расстояния от точки до прямой ij. */
   float maxdist = 0, tmpdist;
   int k, f;

   for (k = i + 1; k < j; k++)
   {
      /* Вычислить расстояние от k-той точки до прямой ij. */
       tmpdist = (
       pts[j].x  *  pts[k].y  +  pts[i].x  *  pts[j].y  +
       pts[k].x  *  pts[i].y  −  pts[j].x  *  pts[i].y  −
       pts[i].x  *  pts[k].y  −  pts[k].x  *  pts[j].y)  /
       sqrt (
       (pts[i].x  −  pts[j].x)  *  (pts[i].x  −  pts[j].x)  +
       (pts[i].y  −  pts[j].y)  *  (pts[i].y  −  pts[j].y));

      /* Модуль расстояния. */
      if (tmpdist < 0) tmpdist = −tmpdist;

      /* Если на данный момент k-тая точка является самой удален-
      ной, то запомнить ее расстояние и номер. */
      if (tmpdist > maxdist)
      {
         maxdist = tmpdist;
         f = k;
      }
   }

   /* Если самая удаленная точка находится на расстоянии, превыша-
   ющем заданную точность приближения, то вернуть f. */
   if (maxdist > e)
      return f;
   /* Иначе — вернуть 0. */
   else
      return 0;
}
```

/* Процедура `readfile` — чтение из файла f координат очередной точки, пока не встретится конец файла. Вызывает функцию `dpbase`, записывает в массив `simp` те точки, которые входят в упрощение, и в массив `pts` все точки, начиная с последней найденной точки поворота.*/

```
void readfile (FILE *f)
{
   int i, k;

   /* Выполнять, пока не встречен конец файла. */
   while (! feof (f))
   {
      /* Считать из файла n-ную точку. */
      fscanf (f, "%f %f\n", &pts [n].x, &pts [n].y);

      if (simpptr == 0)
      {
         simp [simpptr++] = pts [n++];
         continue;
      }

      /* Выполнить упрощение линии. */
      k = dpbase (0, n);

      /* Запомнить все точки, начиная с последней найденной точки
      поворота.*/
      if (k > 0)
      {
         simp [simpptr++] = pts [k];

         for (i = k; i <= n; i++)
            pts [i−k] = pts [i];

         n = n − k;
      }

      n++;
   }

   /* Окончательное упрощение линии. */
   simp [simpptr] = pts [n−1];
}
```

/* Процедура `writefile` — запись точек из массива `simp` в выходной файл f.*/

```
void writefile (FILE *f)
{
    int i;

    /* Вывести точки в файл. */
    for (i = 0; i <= simpptr; i++)
        fprintf (f, "%f %f\n", simp [i].x, simp [i].y);
}

/* Функция main.
Аргументы: первый аргумент в командной строке рассматривается как
имя входного файла, второй как имя выходного файла, третий задает
точность приближения. Входной и выходной файлы представляют собой
пары чисел (координаты точки на плоскости), разделенные переводами
строки. Два числа в паре разделяются пробелом. Точность приближения
— это расстояние до упрощенной линии, которое не должны превышать
точки, не являющиеся точками поворота.
Возвращает: если не было никаких ошибок, то 0; если были ошибки при
открытии файлов, или было получено неправильное число аргументов в
командной строке, то −1. */
int main (int argc, char **argv)
{
    /* Входной и выходной файлы. */
    FILE *fin, *fout;

    /* Если в командной строке не введено имя входного файла имя вы-
    ходного файла, точность приближения линии или введено что-нибудь
    лишнее, то выдать сообщение. */
    if (argc != 4)
    {
        printf ("Usage: %s infile outfile dist\n", argv [0]);
        exit (−1);
    }

    /* Открыть входной файл для чтения. */
    if ((fin = fopen (argv [1], "r")) == NULL)
    {
        printf ("%s: %s: %s\n", argv [0], argv [1], strerror (errno));
        exit (−1);
    }

    /* Открыть выходной файл для записи. */
    if ((fout = fopen (argv [2], "w")) == NULL)
    {
```

```
        printf ("%s: %s: %s\n", argv [0], argv [2], strerror (errno));
        fclose (fin);
        exit (−1);
    }

    /* Запомнить точность приближения. */
    e = atof (argv [3]);

    /* Инициализация указателей. */
    n = 0;
    simpptr = 0;

    /* Выполнить чтение из входного файла с применением алгоритма
    Дугласа—Пекера. */
    readfile (fin);

    /* Записать полученные точки в выходной файл. */
    writefile (fout);

    /* Закрыть файлы. */
    fclose (fin);
    fclose (fout);

    exit (0);
}
```

Список литературы

[1] Hershberger J., Snoeyink J. Cartographic Line Simplification and Polygon CSG Formulae in $O(n\, log^{*} n)$ Time.
http://citeseer.nj.nec.com/hershberger98cartographic.html

[2] Hershberger J., Snoeyink J. Speeding Up the Douglas—Peucer Line Simplification Algorithm.
http://citeseer.nj.nec.com/hershberger92speeding.html

[3] Peter H. Dana. Global Positioning System Overview.
http://www.colorado.edu/geography/gcraft/notes/gps/gps.html