СПб НИУ ИТМО

кафедра ИПМ

Алгоритмы и структуры данных

Лабораторная работа № 2

Основные операции для работы с d-кучами

Нахождение кратчайших путей от вершины s до всех остальных вершин в графе

Работу выполнил:

Студент II курса

Группы № 2120

Журавлев Виталий

Санкт-Петербург

2014 г.

**Цель работы:**

Предлагается попарное сравнение различных алгоритмов нахождения

кратчайших путей от вершины s до всех остальных вершин в графе

G = (V, E), имеющем n вершин и m ребер.

Варианты выбора пары алгоритмов A и B для сравнения:

А - алгоритм Дейкстры, реализованный на основе 5-кучи,

В - алгоритм Дейкстры, реализованный на основе 6-кучи;

Написать программу, реализующую алгоритм А и алгоритм В., для

проведения экспериментов.

Выходом данной программы должно быть время работы ТА алгоритма А и

время работы ТВ алгоритма B в секундах.

Провести эксперименты на основе следующих данных:

n = 1, … ,104+1 с шагом 100, q = 1, r =106, количество ребер: а) m ≈ n2/10, б) m ≈ n2,

нарисовать графики функций TА(n) и ТВ(n) для обоих случаев.

 Сформулировать и обосновать вывод о том, в каких случаях целесообразно применять алгоритм А, а в каких - алгоритм В.

**Код программы:**

#include <iostream>

#include <time.h>

#include <cmath>

#include <algorithm>

#include <vector>

using namespace std;

struct elem

{

 int key;

 int name;

};

struct rebro

{

 int weight;

 int next;

};

class dheap

{

public:

 dheap(vector <int> &v, int d)

 {

 dheap::d = d;

 for (int i = 0; i < v.size(); i++) {

 elem elem1;

 elem1.key = v[i];

 elem1.name = i;

 elements.push\_back(elem1);

 }

 for (int i = 0; i < elements.size(); i++) {

 index.push\_back(i);

 }

 Create\_queue();

 }

 void set\_elem(int i, int x)

 {

 elements[index[i]].key = x;

 elements[index[i]].name = i;

 emersion(index[i]);

 }

private:

 int d;

 vector <elem> elements;

 vector <int> index;

 int minchild(int i)

 {

 int kf = first\_child(i);

 if (kf == 0)

 return i;

 int kl = last\_child(i);

 for (int j = kf + 1; j <= kl; j++) {

 if (elements[j].key < elements[kf].key)

 kf = j;

 }

 return kf;

 }

 int first\_child(int i)

 {

 if ((i - 1)\*d + 2 > elements.size())

 return 0;

 else

 return (i - 1)\*d + 2;

 }

 int last\_child(int i)

 {

 if (first\_child(i) == 0)

 return 0;

 else

 return min(first\_child(i) + d - 1, (int)elements.size() - 1);

 }

 int father(int i)

 {

 if (i - 1 < 0)

 return 0;

 else

 return (i - 2) / d + 1;

 }

 void immersion(int i)

 {

 elem elem0 = elements[i];

 int c = minchild(i);

 while ((c != i) && (elem0.key > elements[c].key)) {

 elements[i] = elements[c];

 index[elements[i].name] = i;

 i = c;

 c = minchild(c);

 }

 elements[i] = elem0;

 index[elements[i].name] = i;

 }

 void emersion(int i)

 {

 elem elem0 = elements[i];

 int p = father(i);

 while (i != 1 && elements[p].key > elem0.key) {

 elements[i] = elements[p];

 index[elements[i].name] = i;

 i = p;

 p = father(p);

 }

 elements[i] = elem0;

 index[elements[i].name] = i;

 }

public: elem extract\_min()

 {

 elem elem1 = elements[0];

 elements[0] = elements[elements.size() - 1];

 index[elements[0].name] = 0;

 elements.pop\_back();

 if (elements.size() > 1)

 immersion(0);

 return elem1;

 }

private:

 void Create\_queue()

 {

 for (int i = elements.size() - 1; i >= 0; i--)

 immersion(i);

 }

};

int main()

{

 int s = 0, r, m, u, v, infinite = INT\_MAX, d = 6;

 clock\_t time;

 freopen("result.csv", "w", stdout);

 printf("N;Time\n");

 for (int n = 1; n <= 3001; n += 100) {

 r = 1000000;

 m = n\*n / 10;

 vector <vector <rebro> > G(n);

 vector <bool> is\_used(n, 0);

 vector <int> a;

 vector <int> dist(n);

 vector <int> pr(n);

 for (int i = 0; i < m; i++) {

 rebro re;

 re.next = rand() % n;

 re.weight = (rand() % r) + 1;

 G[rand() % n].push\_back(re);

 }

 for (int i = 0; i < n; i++) {

 if (!is\_used[i]) {

 is\_used[i] = true;

 a.push\_back(i);

 }

 }

 for (int i = 0; i < a.size() - 1; i++) {

 for (int j = i + 1; j < a.size(); j++) {

 rebro re;

 re.next = a[j];

 re.weight = rand() % r + 1;

 G[a[i]].push\_back(re);

 re.next = a[i];

 G[a[j]].push\_back(re);

 }

 }

 for (int i = 0; i < n; i++)

 dist[i] = infinite;

 if (s < dist.size()) {

 dist[s] = 0;

 }

 else {

 dist[dist.size() - 1] = 0;

 }

 dheap D(dist, d);

 time = clock();

 for (int i = 1; i < n; i++) { //Алгоритм Дейкстры

 elem elem1 = D.extract\_min();

 u = elem1.name;

 for (int j = 0; j < G[u].size(); j++) {

 v = G[u][j].next;

 if (dist[u] + G[u][j].weight < dist[v]) {

 dist[v] = dist[u] + G[u][j].weight;

 pr[v] = u;

 D.set\_elem(v, dist[v]);

 }

 }

 }

 printf("%d;%f\n", n, (double)(clock() - time) / CLOCKS\_PER\_SEC);

 }

 return 0;

}

В переменную d (в методе main) записывается порядок кучи (5 или 6 по варианту). Поэтому алгоритмы A и B одинаковы и различаются только этим порядком.

**Результат вычислений:**

1. m ≈ n2 / 10
2. m ≈ n2

**Вывод:**

В ходе выполнения лабораторной работы был исследован алгоритм Дейкстры на основе d-кучи (A: d = 5 и B: d = 6). Из результата вычислений видно, что время выполнения алгоритма квадратично зависит от количества вершин (O(n^2)).

Так же удалось выяснить, что время выполнения зависит и от количества ребер: чем меньше ребер, тем быстрее выполняется алгоритм.

Из анализа алгоритмов A и B можно сделать такие выводы:

* У алгоритма A более плавная зависимость T от N
* Время выполнения алгоритма A немного меньше времени выполнения B.
* Оба алгоритма работают только с положительным весом ребер.

Применять алгоритм B лучше в случае большого количества вершин, для удобства их хранения. Для небольшого прироста скорости можно применять 5-кучи.