acm.mipt.ru

олимпиады по программированию на Физтехе

Реализация алгоритма Крускала построения минимального остовного дерева на C

- Описание идеи алгоритма

Содержание

Пстановка задачи

Дан связный граф с взвешенными ребрами. Задача заключается в том, чтобы найти подмножество ребер минимального суммарного веса, которые бы покрывали все вершины графа. Другими словами, необходимо удалить как можно больше ребер (по весу) сохранив при этом свойство связности графа.

Формат входа

Вершины считаются пронумерованными от 0 и до V-1. Вход состоит из E+1 строчек. В первой даны V и E – количество вершин и количество ребер. Затем идет E строчек с описанием ребер:

```
V  E
A1  B1  W1
A2  B2  W2
 ....
AE  BE  WE
```

Где (Ai, Bi) – i-ое ребро, Wi – вес этого ребра.

Для работы с вершинами, которые задаются, например, *строковыми именами*, следует обратить внимание на хеширование.

```
Sample input #1
5 8
0 3 5
0 1 1
0 2 3
2 1 6
2 3 1
4 1 3
4 2 2
4 3 2

Sample output #1
2 3 1
0 1 1
4 2 2
0 2 3
```

Первый вариант реализации

```
#include <stdio.h>
#include <stdlib.h>
```

Поиск
Поиск
Раздел «Алгоритмы»
Главная
Форум
Ссылки
El Judge
Инструменты:
Поиск
Изменения
Index
Статистика
Разделы
Информация
Алгоритмы
Язык Си
Язык Ruby
Язык Ассемблера
El Judge
Парадигмы
Образование
Сети
Objective C
Logon>>

```
int NV;                  // Количество вершин в графе
int NE;                  // Количество ребер в графе

#define MAX_NODES 100  // Максимальное количество вершин
#define MAX_EDGES 10   // Максимальное количество ребер в графе

struct edge_t {
   int n1,n2;  // направление
   int w;      // вес ребра
} edges[MAX_EDGES];  // Ребра графа

int nodes[MAX_NODES];  // Вершины графа.  Значение - "верхняя вершина"

// Функция "сравнения" двух ребер, используемая для сортировки
int cmp(const void *a,const void *b){
    edge *c=(edge*)a, *d=(edge*)b;
   return c->w - d->w;
}

int last_n;

// Функция получает цвет вершины n-й по порядку.
// если nodes[n] < 0, то вершина n имеет цвет nodes[n]
// если nodes[n] >= 0, то вершина n имеет цвет такой же,
// как и вершина с номером nodes[n]
int getColor(int n){
   int c;
   if (nodes[n]<0)
      return nodes[last_n=n];
   c = getColor(nodes[n]);
   nodes[n] = last_n;
   return c;
}

int main(){
   int i;
   // Считываем вход
   scanf ("%d %d", &NV, &NE);
   for(i = 0; i < N; i++) nodes[i] = -1-i;

   for(i = 0; i < NE; i++)
      scanf("%d %d %d", &edges[i].n1, &edges[i].n2, &edges[i].w);

   // Алгоритм Крускала

   // Сортируем все ребра в порядке возрастания весов
   qsort(edges, NE, sizeof(edge_t), cmp);

   for(i = 0; i < NE; i++){ // пока не прошли все ребра
      int c2 = getColor(edges[i].n2);
      if ( getColor (edges[i].n1) != c2 ){
         // Если ребро соединяет вершины различных цветов-мы его добавляем
         // и перекрашиваем вершины
         nodes [last_n] = edges[i].n2;
         printf ("%d %d %d\n", edges[i].n1, edges[i].n2, edges[i].w);
      }
   }
   return 0;
}
```

-- VladimirSitnikov - 03 Apr 2004

Второй вариант реализации

Формат входа у этой программы такой же, как и у предыдущей.

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
```

```
typedef struct {
    int from;       // edge start vertex
    int to;         // edge end vertex
    double w;       // edge weight
} edge_t;

typedef struct set_t {
    struct set_t *p; // link on parent
} set_t;


int NS;      // number of sets
set_t *sets; // array of sets
int NE;      // number of edges
edge_t *E;   // array of edges
int NV;      // number of edges

// compare function for sorting edges by weight
int cmpw(edge_t *a, edge_t *b)
{
    if(a->w > b->w ) return 1;
    if(a->w < b->w ) return -1;
    return 0;
}

set_t*
get_set_id(set_t* s)
{
    if(s == s->p )
        return s;
    else {
        set_t *p = get_set_id(s->p);
        s->p = p;
        return p;
    }
}

set_t*
join_sets(set_t *a, set_t *b)
{
    a->p = b;
    return a;
}


void
take_edge(int edge_id)
{
    printf("%d %d %lf\n", E[edge_id].from, E[edge_id].to, E[edge_id].w);
}

int
main()
{
    int i;
    double W = 0;
    scanf("%d%d", &NV, &NE);
    E = (edge_t*) malloc(NE * sizeof(edge_t));
    sets = (set_t*) malloc(NV * sizeof(set_t));
    for(i = 0; i < NE ; i++)
    {
        scanf("%d%d%lf", &E[i].from, &E[i].to, &E[i].w);
    }

    // Sort edges by weight
    qsort(E, NE, sizeof(edge_t), (int (*)(const void*, const void*)) cmpw);

    // Create set of one-point sets
```

```
    NS = NV;
    for(i = 0; i < NS ; i++)
        sets[i].p = &sets[i];

    // Extract next edge with minumum weight
    for(i=0; NS > 1 && i < NE; i++)
    {
        // if the edge can't be added to tree, then go o next edge
        if ( get_set_id ( &sets[E[i].from]) == get_set_id ( &sets[E[i].to]) )
                continue;
        // add the edge to covering tree
        join_sets ( get_set_id (&sets[E[i].from] ),  get_set_id ( &sets[E[i].to]) );
        NS--;
        take_edge(i);
        W += E[i].w;
    }

    if(NS != 1)
       fprintf(stderr, "warning: Graph is not connected.\n");
    printf("Covering tree weight = %lf\n", W);
}
```

-- ArtemVoroztsov - 16 Mar 2005