Example of Kurskay’s Algorithm using Disjoint Sets and Heap

acarlstein.com/

Posted by Alejandro G. Carlstein Ramos Mejia on October 15, 2010 November 2, 2010 About Programming / Algorithms / ANSI/POSIX C

NOTIFICATION: These examples are provided for educational purposes. The use of this code and/or information is under your own responsibility and risk. The information and/or code is given ‘as is’. I do not take responsibilities of how they are used.

Example of Kurskay’s algorithm using disjoint sets and heap.

kurskay_djoinset_heap.c:

```
/*
 * Program: 08
 * Author: Alejandro G. Carlstein
 * Description: Applying Kurskay's Algorithm using Disjoint Sets
 */

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

#define MAX_EDGES 100001
#define MAX_VERTICES  500001
#define MAX_HEAP 100001
#define LEFT(i) ((i << 1) + 1)
#define RIGHT(i) ((i << 1) + 2)
#define PARENT(i) (((i + 1) >> 1) - 1)
#define DIV_BY_2(i) (i >> 1)
#define ROOT_INDEX 0

#define DBG_LV0 0
#define DBG_LV1 0
#define DBG_LV2 0
#define DBG_LV3 0

struct Vertex{
 int id;
 int rank;
 struct Vertex *parent;
};

struct Edge{
  struct Vertex *startVertex;
  struct Vertex *endVertex;
  int length;
```

```
};

struct Edge edges[MAX_EDGES];
struct Vertex vertices[MAX_VERTICES];
int heap[MAX_EDGES];

void scanEdges(void);
void debug(const char *message, int isDebugging);
void printEdges(struct Edge edges[], int length);
void initHeap(void);
int *mstKruskal(int *numEdges);
void printVertices(struct Vertex vertices[]);
void heapSort(int array[], int length);
void buildMinHeap(int array[], int heapSize);
void minHeapify(int array[], int index, int heapSize);
void exchange(int *a, int *b);
void printArray(int array[], int length);
void makeSet(struct Vertex *vertex);
struct Vertex* findSet(struct Vertex *vertex);
void unionTrees(struct Vertex *vertexX, struct Vertex *vertexY);
void link(struct Vertex *vertexX, struct Vertex *vertexY);
void printEdgesUsingHeapArray(int heapArray[], int length);
void printWeightOfTree(int heapArray[], int length);

int numVertices;
int numEdges;

int main(int argc, char* argv[]){

 scanEdges();
 initHeap();
 int length = numEdges;
  int *arrayHeapEdges;
  arrayHeapEdges = mstKruskal(&length);

  if (DBG_LV1) printArray(arrayHeapEdges, length);

  if(DBG_LV1) printVertices(vertices);

 if (DBG_LV1) printEdgesUsingHeapArray(arrayHeapEdges, length);

 printWeightOfTree(arrayHeapEdges, length);

 free(arrayHeapEdges);

 return 0;

}

void scanEdges(void){
 debug('scanEdges()', DBG_LV0);

 scanf('%d %d', &numVertices, &numEdges);
 if (DBG_LV1) printf('numVertices: %d, numEdges: %d\n', numVertices, numEdges);

 int i;
 int startVertexId = -1;
 int endVertexId = -1;
 for (i = 0;
    scanf('%d %d %d', &startVertexId, &endVertexId, &edges[i].length) == 1 ||
        i < numEdges;
    ++i){

  if (DBG_LV2)
   printf('startVertexId: %d, endVertexId: %d\n', startVertexId, endVertexId);

  if (startVertexId >= numVertices ||
    endVertexId >= numVertices){
   fprintf(stderr, 'Error: Vertex id is outside the maximum range of vertices\n');
   exit(1);
  }

  vertices[startVertexId].id = startVertexId;
  edges[i].startVertex = &vertices[startVertexId];

  vertices[endVertexId].id = endVertexId;
  edges[i].endVertex = &vertices[endVertexId];

  if(DBG_LV2)
   printf('edges[%d].startVertex->id: %d, edges[%d].endVertex->id: %d\n',
       i, edges[i].startVertex->id, i, edges[i].endVertex->id);

 }

 if (DBG_LV1)
  printEdges(edges, numEdges);

}

void debug(const char *message, int isDebugging){
 if (isDebugging) printf('%s\n', message);
}

void printEdges(struct Edge edges[], int length){
 debug('printEdges()', DBG_LV0);

 printf('Edges: \n');
 int i;
 for (i = 0; i < length; ++i)
  printf('%d(%d <%d> %d) ',
      i, edges[i].startVertex->id, edges[i].length, edges[i].endVertex->id);

  printf('\n');
}
```

```
void initHeap(void){
 debug('initHeap()', DBG_LV0);

 int i;

 for (i = 0; i < numEdges; ++i)
  heap[i] = i;
}

int *mstKruskal(int *numEdges){
 debug('mstKruskal()', DBG_LV0);

 // A <- 0
 int *arrayEdgesIndex = (int*) malloc(*numEdges * sizeof(int));

 int vertexIndex;
 for (vertexIndex = 0; vertexIndex < numVertices; ++vertexIndex)
  makeSet(&vertices[vertexIndex]);

 if (DBG_LV1) printVertices(vertices);

 // Sort edges into nondecreasing order by weight
 heapSort(heap, *numEdges);

 int index = 0;
 int edgeIndex;
 for (edgeIndex = 0; edgeIndex < *numEdges; ++edgeIndex){

  if (DBG_LV2)
   printf('V: %d\n', (int)findSet(edges[heap[edgeIndex]].startVertex));

  if ((int)findSet(edges[heap[edgeIndex]].startVertex) !=
    (int)findSet(edges[heap[edgeIndex]].endVertex)  ){

   if (DBG_LV2)
    printf('findset(edges[%d]) = findset(edges[%d])\n',
        heap[edgeIndex], heap[edgeIndex]);

   // A <- A U {(u,v)}
   arrayEdgesIndex[index++] = heap[edgeIndex];

   unionTrees(edges[heap[edgeIndex]].startVertex,
         edges[heap[edgeIndex]].endVertex);
  }

 }

 *numEdges = index;

 if (DBG_LV1) printArray(arrayEdgesIndex, index);

 return (arrayEdgesIndex);
}
```

```
void printVertices(struct Vertex vertices[]){
 debug('printVertices()', DBG_LV0);

 int i;
 for (i = 0; i < numVertices; ++i)
  printf('(%d)[%d] Id: %d, Rank: %d, *Parent:%d \n',
      (int)&vertices[i], i, vertices[i].id, vertices[i].rank, (int)vertices[i].parent);

 printf('\n');
}

void heapSort(int array[], int length){
 if (DBG_LV0) printf('heapSort(length: %d)\n', length);

 buildMinHeap(array, length);

 int i;
 int heap_size = length;
 for (i = length - 1 ; i >= 0; --i){
  exchange(&array[0], &array[i]);
  heap_size--;
  minHeapify(array, 0, heap_size);
 }

}

void buildMinHeap(int array[], int heapSize){
 debug('buildMinHeap', DBG_LV0);

 int index;
 for (index = DIV_BY_2(heapSize); index >= ROOT_INDEX; --index)
  minHeapify(array, index, heapSize);
}

void minHeapify(int array[], int index, int heapSize){
 debug('minHeapify()', DBG_LV0);

 int left, right, smallest;

 smallest = index;
 left = LEFT(index);
 right = RIGHT(index);

 if (left < heapSize &&
    edges[array[left]].length > edges[array[index]].length)
  smallest = left;

 if (right < heapSize &&
    edges[array[right]].length > edges[array[smallest]].length)
  smallest = right;

 if (smallest != index){
  exchange(&array[index], &array[smallest]);
  minHeapify(array, smallest, heapSize);
 }

}

void exchange(int *a, int *b){
 debug('exchange()', DBG_LV3);

 int temp;
 temp = *a;
 *a = *b;
 *b = temp;
}

void makeSet(struct Vertex *vertex){
 debug('makeSet()', DBG_LV0);

 vertex->parent = vertex;
 vertex->rank = 0;
}

struct Vertex* findSet(struct Vertex *vertex){
 debug('findSet()', DBG_LV0);

 if (vertex == NULL)
  debug('Vertex is NULL', DBG_LV1);

 if (vertex != vertex->parent){
  vertex->parent = findSet(vertex->parent);
 }

 return (vertex->parent);
}

void unionTrees(struct Vertex *vertexX, struct Vertex *vertexY){
 debug('unionTrees()', DBG_LV0);

 link(findSet(vertexX), findSet(vertexY));
}

void link(struct Vertex *vertexX, struct Vertex *vertexY){
 debug('link()', DBG_LV0);

 if (vertexX->rank > vertexY->rank){
  vertexY->parent = vertexX;
 }else{
  vertexX->parent = vertexY;
  if (vertexX->rank == vertexY->rank)
   ++vertexY->rank;
 }

}
```

```
void printArray(int array[], int length){
 debug('printArray()', DBG_LV0);

 int i;

 for (i = 0; i < length; ++i)
  printf('[%d]', i);

 printf('\n');

 for (i = 0; i < length; ++i)
  printf(' %d ', array[i]);

 if (DBG_LV1) printf('\n\n');
}

void printEdgesUsingHeapArray(int heapArray[], int length){
 debug('printEdgesUsingHeapArray()', DBG_LV0);

 int i;
 for (i = 0; i < length; ++i)
  printf('%d-%d(%d) ',
      edges[heapArray[i]].startVertex->id, edges[heapArray[i]].endVertex->id, (i + 1));
}

void printWeightOfTree(int heapArray[], int length){
 debug('printWeightOfTree()', DBG_LV0);

 int result = 0;
 int i;
 for (i = 0; i < length; ++i)
  result += edges[heapArray[i]].length;

 printf('%d\n', result);

}
```

input.txt:

```
4 7
0 1 5
0 2 4
1 3 3
2 3 2
3 0 1
2 1 10
0 3 20
```

If you encounter any problems or errors, please let me know by providing an example of the code, input, output, and an explanation. Thanks.