Example of Vertex Cover Algorithm using Heap

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 Vertex Cover algorithm using heap.

vertex_cover.c:

```
/*
 * Program: 10
 * Author: Alejandro G. Carlstein
 * Description: Applying Vertex Cover Algorithm
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <errno.h>
#include <string.h>

#define MAX_EDGES 100001
#define MAX_VERTICES  50001
#define LEFT(i) ((i << 1) + 1)
#define RIGHT(i) ((i << 1) + 2)
#define DIV_BY_2(i) (i >> 1)
#define FALSE 0
#define TRUE 1

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

struct Edge{
 int id;
  int uVertex;
  int vVertex;
  short enabled;
} edges[MAX_EDGES];

struct Vertex{
 int id;
 int numEdges;
}vertices[MAX_VERTICES];
```

```
struct VertexCover{
 struct Vertex *vertexPointer;
 struct VertexCover *next;
};

struct AdjList{
 struct Edge *edgePointer;
 struct AdjList *next;
} *adjList[MAX_EDGES], *headAdj, *newAdj;

void readLimits(void);
void setVerticesDefault(void);
void readEdges(void);
void printEdges(void);
void printVertices(void);
struct VertexCover *vertexCover(void);
void makeMaxPriorityQueue(int heapArray[], int *length);
void printArray(int array[], int length);
void buildMaxHeap(int heapArray[], int length);
void maxHeapify(int heapArray[], int index, int heapSize);
int getMaxHeap(int heapArray[], int heapSize);
int heapExtractMax(int heapArray[], int *heapSize);
void exchange(int *a, int *b);
void buildAdjacentEdgesList(void);
void insertAdjEdgesOf(int edgeIndex);
void insertAdjEdge(int edgeIndex, int adjEdgeIndex);
void printAdjList(int edgeIndex);
void addVerticesOfEdgeToVertexCover(struct VertexCover *vertexCover, int vertexIndex);
void printvertexCover(struct VertexCover *vertexCover);
void debug(int debugLevel, char *fmt, ...);
void errorDoExit(char *fmt, ...);

int numVertices, numEdges;
int main(int argc, char *argv[]){
 readLimits();
 setVerticesDefault();
 readEdges();
 printvertexCover(vertexCover());
 return 0;
}

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

 scanf('%d %d', &numVertices, &numEdges);
 debug(DBG_LV1, '# of vertices: %d, # of edges: %d', numVertices, numEdges);
}

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

 int i;
```

```
 for (i = 0; i < numVertices; ++i){
  vertices[i].id = i;
  vertices[i].numEdges = 0;
 }
}

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

 int i, uVertex, vVertex;
 for(i = 0; i < numEdges; ++i){
  int length;
  scanf('%d %d %d', &uVertex, &vVertex, &length);
  if (uVertex >= numVertices || vVertex >= numVertices)
   errorDoExit('Edge [%d](%d <> %d) have vertices id > %d', i, uVertex, vVertex,
numVertices);
  edges[i].id = i;
  edges[i].uVertex = uVertex;
  edges[i].vVertex = vVertex;
  edges[i].enabled = TRUE;
  ++vertices[uVertex].numEdges;
  ++vertices[vVertex].numEdges;
 }

 if (DBG_LV1) printEdges();
 if (DBG_LV1) printVertices();
}

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

 int i;
 for(i = 0; i < numEdges; ++i)
  if (edges[i].enabled){
   printf('(%d<[%d]>%d)\n', edges[i].uVertex, i,edges[i].vVertex);
  }else{
   printf('(%d|[%d]|%d)\n', edges[i].uVertex, i,edges[i].vVertex);
  }
}

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

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

 printf('\n[NE]');
 for (i = 0; i < numVertices; ++i)
  printf(' %d ' , vertices[i].numEdges);
 printf('\n');
}
```

```
struct VertexCover *vertexCover(void){
 debug(DBG_LV0, 'vertexCover()');

 // Build list of vertices that have the most number of edges
 int heapLength = numEdges + 1;
 int heapArray[heapLength];
 makeMaxPriorityQueue(heapArray, &heapLength);
 if (DBG_LV1) printArray(heapArray, heapLength);

 // Build Adjacent Vertices List
 buildAdjacentEdgesList();

 // Vertex cover to being constructed
 struct VertexCover *vertexCover = NULL;

 int edgeIndex;
 while(heapLength > 0){
  // Pick the edge in which vertices have the highest number of edges
  edgeIndex = heapExtractMax(heapArray, &heapLength);
  //Keep looping until you find an edge that is enabled or your count all the edges
  if (edges[edgeIndex].enabled){
   debug(DBG_LV1, 'EDGE FOUND: (%d<[%d]>%d)', edges[edgeIndex].uVertex, edgeIndex,
edges[edgeIndex].vVertex);
  }else{
   debug(DBG_LV1, 'EDGE FOUND: (%d<|%d|>%d)', edges[edgeIndex].uVertex, edgeIndex,
edges[edgeIndex].vVertex);
  }

  if (edges[edgeIndex].enabled){

   int uVertex = edges[edgeIndex].uVertex;
   int vVertex = edges[edgeIndex].vVertex;
   int uVertexNumEdges = vertices[uVertex].numEdges;
   int vVertexNumEdges = vertices[vVertex].numEdges;
   debug(DBG_LV1,'uVertex: %d, NumEdges: %d', uVertex, uVertexNumEdges);
   debug(DBG_LV1,'vVertex: %d, NumEdges: %d', vVertex, vVertexNumEdges);

   // Add this vertices of this edge to the list of edges
   struct VertexCover *newVertexCover = (struct VertexCover *) malloc(sizeof(struct
VertexCover));
   newVertexCover->vertexPointer = &vertices[uVertex];
   if (vertexCover == NULL){
    debug(DBG_LV1, 'vertexCover is empty');
    newVertexCover->next = vertexCover;
    vertexCover = newVertexCover;
   }else{
    debug(DBG_LV1, 'vertexCover is NOT empty');
    struct VertexCover *currentVertexCover = vertexCover;
    while (currentVertexCover->next != NULL){
     currentVertexCover = currentVertexCover->next;
    }
    newVertexCover->next = currentVertexCover->next;
```

```
    currentVertexCover->next = newVertexCover;
   }
   debug(DBG_LV1, 'adding %d to vertexCover', newVertexCover->vertexPointer->id);

   // Add this vertices of this edge to the list of edges
   newVertexCover = (struct VertexCover *) malloc(sizeof(struct VertexCover));
   newVertexCover->vertexPointer = &vertices[vVertex];
   if (vertexCover == NULL){
    debug(DBG_LV1, 'vertexCover is empty');
    newVertexCover->next = vertexCover;
    vertexCover = newVertexCover;
   }else{
    debug(DBG_LV1, 'vertexCover is NOT empty');
    struct VertexCover *currentVertexCover = vertexCover;
    while (currentVertexCover->next != NULL){
     currentVertexCover = currentVertexCover->next;
    }
    newVertexCover->next = currentVertexCover->next;
    currentVertexCover->next = newVertexCover;
   }
   debug(DBG_LV1, 'adding %d to vertexCover', newVertexCover->vertexPointer->id);

   // Search thought all the adjacent edges to this edge and disable them
   struct AdjList *tempAdj;
   tempAdj = adjList[edgeIndex];
   while(tempAdj != NULL){
     debug(DBG_LV1, 'Disable Edge (%d<|%d|>%d)',
        tempAdj->edgePointer->uVertex, tempAdj->edgePointer->id, tempAdj->edgePointer-
>vVertex);
     tempAdj->edgePointer->enabled = FALSE;
     vertices[tempAdj->edgePointer->uVertex].numEdges--;
     vertices[tempAdj->edgePointer->vVertex].numEdges--;
    tempAdj = tempAdj->next;
   }
   edges[edgeIndex].enabled = FALSE;
   vertices[edges[edgeIndex].uVertex].numEdges--;
   vertices[edges[edgeIndex].vVertex].numEdges--;

  }
 }
 if(DBG_LV1) printEdges();

 return vertexCover;
}

void makeMaxPriorityQueue(int heapArray[], int *length){
 debug(DBG_LV0, 'makePriorityQueue(length: %d)', *length);

 int i;
 for (i = 0; i < numEdges; ++i)
  heapArray[i] = i;

 *length = numEdges;
```

```
 if (DBG_LV1) printArray(heapArray, *length);
 buildMaxHeap(heapArray, *length);
 if (DBG_LV1) printArray(heapArray, *length);
 if (DBG_LV1) printEdges();
}

void printArray(int array[], int length){
 debug(DBG_LV0, 'printArray(length: %d)', length);

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

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

void buildMaxHeap(int heapArray[], int length){
 debug(DBG_LV0, 'buildMaxHeap(length: %d)', length);

 int heapSize = length;

 int index;
 for (index = DIV_BY_2(length); index > -1; --index)
  maxHeapify(heapArray, index, heapSize);
}

void maxHeapify(int heapArray[], int index, int heapSize){
 debug(DBG_LV2, 'maxHeapify(index: %d, heapSize: %d)', index, heapSize);

 int largestIndex = index;
 int leftIndex = LEFT(index);
 int rightIndex = RIGHT(index);
 debug(DBG_LV2, '-LargestIndex: %d, leftIndex: %d, rightIndex: %d', largestIndex,
leftIndex, rightIndex);

 if (leftIndex <= heapSize && rightIndex <= heapSize){

  unsigned int indexValue = vertices[edges[heapArray[index]].uVertex].numEdges +
              vertices[edges[heapArray[index]].vVertex].numEdges;

  unsigned long int leftValue = vertices[edges[heapArray[leftIndex]].uVertex].numEdges
+
              vertices[edges[heapArray[leftIndex]].vVertex].numEdges;

  if ((leftIndex < heapSize) && (leftValue > indexValue))
   largestIndex = leftIndex;
  debug(DBG_LV2, '(left) largestIndex: %d', largestIndex);

  debug(DBG_LV2, 'rightIndex: %d', rightIndex);
  debug(DBG_LV2, 'edges[%d].uVertex: %d, edges[%d].vVertex: %d',
```

```
          edges[heapArray[rightIndex]].uVertex,
          edges[heapArray[rightIndex]].vVertex);
  debug(DBG_LV2, 'uVertexNum: %d, vVertexNum: %d',
         vertices[edges[heapArray[rightIndex]].uVertex].numEdges,
         vertices[edges[heapArray[rightIndex]].vVertex].numEdges);
  unsigned int rightValue = vertices[edges[heapArray[rightIndex]].uVertex].numEdges +
               vertices[edges[heapArray[rightIndex]].vVertex].numEdges;

  if ((rightIndex < heapSize) && (rightValue > indexValue))
   largestIndex = rightIndex;
  debug(DBG_LV2, '(right) largestIndex: %d', largestIndex);

  if (largestIndex != index){
   debug(DBG_LV2, 'largestIndex != index');
   exchange(&heapArray[index], &heapArray[largestIndex]);
   maxHeapify(heapArray, largestIndex, heapSize);
  }
 }
}

int getMaxHeap(int heapArray[], int heapSize){
 debug(DBG_LV0, 'getMaxHeap(heapSize: %d)', heapSize);

 if (heapSize < 1)
  errorDoExit('Heap Underflow');
 int max = heapArray[0];
 return max;
}

int heapExtractMax(int heapArray[], int *heapSize){
 debug(DBG_LV0, 'heapExtractMax(heapSize: %d)', *heapSize);

 if (*heapSize < 1)
  errorDoExit('Heap Underflow');

 buildMaxHeap(heapArray, *heapSize);

 int max = heapArray[0];
 --*heapSize;
 heapArray[0] = heapArray[*heapSize];
 maxHeapify(heapArray, 1, *heapSize);

 return max;
}

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

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

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

 int edgeIndex;
 for (edgeIndex = 0; edgeIndex < numEdges; ++edgeIndex){
  insertAdjEdgesOf(edgeIndex);
 }

 if (DBG_LV1)
  for (edgeIndex = 0; edgeIndex < numEdges; ++edgeIndex)
   printAdjList(edgeIndex);
}

void insertAdjEdgesOf(int edgeIndex){
 debug(DBG_LV0, 'insertAdjEdgesOf(edgeIndex: %d)', edgeIndex);

 debug(DBG_LV1, ' Searching for adjacent Edges');
 int adjEdgeIndex;
 for (adjEdgeIndex = 0; adjEdgeIndex < numEdges; ++adjEdgeIndex){
  if (edgeIndex != adjEdgeIndex){
   int uVertex = edges[edgeIndex].uVertex;
   int vVertex = edges[edgeIndex].vVertex;
   int adjUVertex = edges[adjEdgeIndex].uVertex;
   int adjVVertex = edges[adjEdgeIndex].vVertex;
   debug(DBG_LV1, '  (%d <[%d]> %d) <=?=> A(%d <[%d]> %d)', uVertex, edgeIndex,
vVertex, adjUVertex, adjEdgeIndex, adjVVertex);
   if (uVertex == adjUVertex || vVertex == adjVVertex ||
     uVertex == adjVVertex || vVertex == adjUVertex){
    debug(DBG_LV1, '   YES');
    insertAdjEdge(edgeIndex, adjEdgeIndex);
   }
  }
 }
 if(DBG_LV1) printAdjList(edgeIndex);
}

void insertAdjEdge(int edgeIndex, int adjEdgeIndex){

 struct AdjList *headAdj;
 headAdj = adjList[edgeIndex];

 struct AdjList *newAdj = (struct AdjList *) malloc(sizeof(struct AdjList));
 newAdj->edgePointer = &edges[adjEdgeIndex];
 newAdj->next = NULL;

 if (headAdj == NULL){
  newAdj->next = adjList[edgeIndex];
  adjList[edgeIndex] = newAdj;
 }else{
  struct AdjList *currentAdj = adjList[edgeIndex];
  while (currentAdj->next != NULL)
   currentAdj = currentAdj->next;
```

```
  newAdj->next = currentAdj->next;
  currentAdj->next = newAdj;
 }
}

void printAdjList(int edgeIndex){
 debug(DBG_LV0, '**printAdjList(edgeIndex: %d)', edgeIndex);

 struct AdjList *tempAdj;
 tempAdj = adjList[edgeIndex];

 if (tempAdj == NULL){
  debug(DBG_LV1, ' tempAdj is empty');
 }else{

  printf('[%d]', edgeIndex);
  int i = 0;
  while(tempAdj != NULL){
   printf('|%d|', i++);
   tempAdj = tempAdj->next;
  }

  printf('\n i:');
  tempAdj = adjList[edgeIndex];
  while(tempAdj != NULL){
   printf('[%d]', tempAdj->edgePointer->id);
   tempAdj = tempAdj->next;
  }

  printf('\n u:');
  tempAdj = adjList[edgeIndex];
  while(tempAdj != NULL){
    printf(' %d ' , tempAdj->edgePointer->uVertex);
    tempAdj = tempAdj->next;
  }

  printf('\n v:');
  tempAdj = adjList[edgeIndex];
  while(tempAdj != NULL){
    printf(' %d ' , tempAdj->edgePointer->vVertex);
    tempAdj = tempAdj->next;
  }

  printf('\n');
 }
}

void addVerticesOfEdgeToVertexCover(struct VertexCover *vertexCover, int vertexIndex){
 debug(DBG_LV0, 'addVerticesOfEdgeToVertexCover(vertexIndex: %d)', vertexIndex);

 struct VertexCover *newVertexCover = (struct VertexCover *) malloc(sizeof(struct
VertexCover));
 newVertexCover->vertexPointer = &vertices[vertexIndex];
```

```
 if (vertexCover == NULL){
  debug(DBG_LV1, 'vertexCover is empty');
  newVertexCover->next = vertexCover;
  vertexCover = newVertexCover;
  debug(DBG_LV1, 'adding %d to vertexCover', newVertexCover->vertexPointer->id);
 }else{
  struct VertexCover *currentVertexCover = vertexCover;
  debug(DBG_LV1, 'adding %d to vertexCover', currentVertexCover->vertexPointer->id);
  while (currentVertexCover->next != NULL){
   debug(DBG_LV1, 'vertexId: %d', currentVertexCover->vertexPointer->id);
   currentVertexCover = currentVertexCover->next;
  }
  newVertexCover->next = currentVertexCover->next;
  currentVertexCover->next = newVertexCover;
 }

}

void printvertexCover(struct VertexCover *vertexCover){
 debug(DBG_LV0, 'printvertexCover');

 struct VertexCover *tempAdj;
 tempAdj = vertexCover;

 if (tempAdj == NULL){
  debug(DBG_LV1, ' tempAdj is empty');
 }else{

  while(tempAdj != NULL){
   printf('%d ', tempAdj->vertexPointer->id);
   tempAdj = tempAdj->next;
  }

  printf('\n');
 }
}

void debug(int debugLevel, char *fmt, ...){
 if (debugLevel == 1){
  va_list argp;
  fprintf(stdout, '[DBG] ');
  va_start(argp, fmt);
  vfprintf(stdout, fmt, argp);
  va_end(argp);
  fprintf(stdout, '\n');
 }
}

void errorDoExit(char *fmt, ...){
 va_list argp;
 fprintf(stderr, '[Error] ');
 va_start(argp, fmt);
 vfprintf(stderr, fmt, argp);
```

```
 va_end(argp);
 if (errno){
  fprintf(stderr, '=> %s\n', strerror(errno));
 }else{
  fprintf(stderr, '\n');
 }
 exit(1);
}
```

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.