Greedy Algorithms | Set 6 (Prim's MST for Adjacency List Representation)

We recommend to read following two posts as a prerequisite of this post.

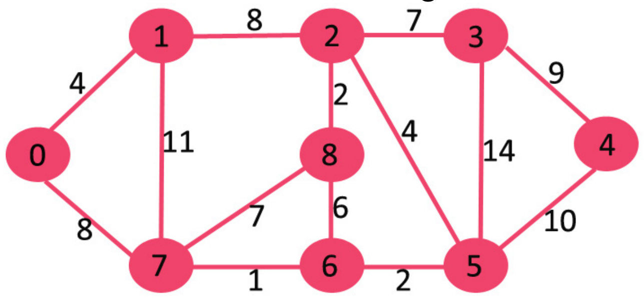
1. Greedy Algorithms | Set 5 (Prim's Minimum Spanning Tree (MST))
2. Graph and its representations

We have discussed Prim's algorithm and its implementation for adjacency matrix representation of graphs. The time complexity for the matrix representation is O(V^2). In this post, O(ELogV) algorithm for adjacency list representation is discussed.
As discussed in the previous post, in Prim's algorithm, two sets are maintained, one set contains list of vertices already included in MST, other set contains vertices not yet included. With adjacency list

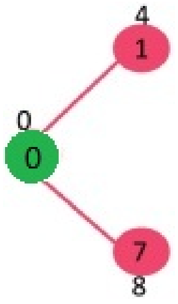
representation, all vertices of a graph can be traversed in O(V+E) time using BFS. The idea is to traverse all vertices of graph using BFS and use a Min Heap to store the vertices not yet included in MST. Min Heap is used as a priority queue to get the minimum weight edge from the cut. Min Heap is used as time complexity of operations like extracting minimum element and decreasing key value is O(LogV) in Min Heap.

Following are the detailed steps.
1) Create a Min Heap of size V where V is the number of vertices in the given graph. Every node of min heap contains vertex number and key value of the vertex.
2) Initialize Min Heap with first vertex as root (the key value assigned to first vertex is 0). The key value assigned to all other vertices is INF (infinite).
3) While Min Heap is not empty, do following
…..**a)** Extract the min value node from Min Heap. Let the extracted vertex be u.
…..**b)** For every adjacent vertex v of u, check if v is in Min Heap (not yet included in MST). If v is in Min Heap and its key value is more than weight of u-v, then update the key value of v as weight of u-v.

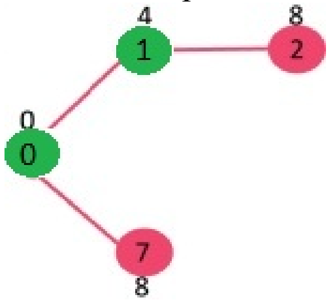
Let us understand the above algorithm with the following example:



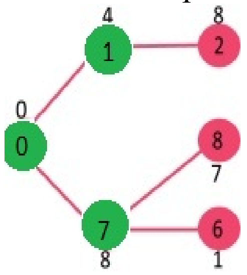
Initially, key value of first vertex is 0 and INF (infinite) for all other vertices. So vertex 0 is extracted from Min Heap and key values of vertices adjacent to 0 (1 and 7) are updated. Min Heap contains all vertices except vertex 0.
The vertices in green color are the vertices included in MST.



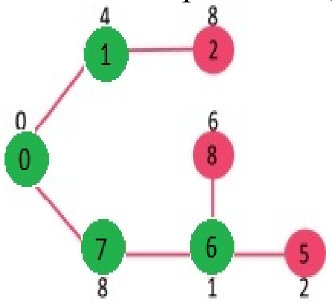
Since key value of vertex 1 is minimum among all nodes in Min Heap, it is extracted from Min Heap and key values of vertices adjacent to 1 are updated (Key is updated if the a vertex is not in Min Heap and previous key value is greater than the weight of edge from 1 to the adjacent). Min Heap contains all vertices except vertex 0 and 1.



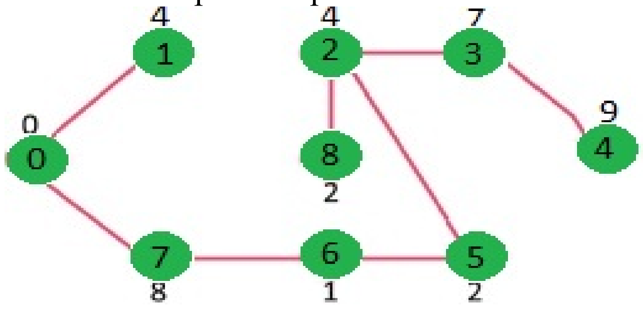
Since key value of vertex 7 is minimum among all nodes in Min Heap, it is extracted from Min Heap and key values of vertices adjacent to 7 are updated (Key is updated if the a vertex is not in Min Heap and previous key value is greater than the weight of edge from 7 to the adjacent). Min Heap contains all vertices except vertex 0, 1 and 7.



Since key value of vertex 6 is minimum among all nodes in Min Heap, it is extracted from Min Heap and key values of vertices adjacent to 6 are updated (Key is updated if the a vertex is not in Min Heap and previous key value is greater than the weight of edge from 6 to the adjacent). Min Heap contains all vertices except vertex 0, 1, 7 and 6.



The above steps are repeated for rest of the nodes in Min Heap till Min Heap becomes empty



```
// C / C++ program for Prim's MST for adjacency list representation of graph

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

// A structure to represent a node in adjacency list
struct AdjListNode
{
    int dest;
    int weight;
    struct AdjListNode* next;
};

// A structure to represent an adjacency liat
struct AdjList
{
    struct AdjListNode *head;  // pointer to head node of list
};

// A structure to represent a graph. A graph is an array of adjacency lists.
// Size of array will be V (number of vertices in graph)
struct Graph
{
    int V;
    struct AdjList* array;
};

// A utility function to create a new adjacency list node
struct AdjListNode* newAdjListNode(int dest, int weight)
{
    struct AdjListNode* newNode =
            (struct AdjListNode*) malloc(sizeof(struct AdjListNode));
    newNode->dest = dest;
    newNode->weight = weight;
    newNode->next = NULL;
    return newNode;
}

// A utility function that creates a graph of V vertices
struct Graph* createGraph(int V)
{
    struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
    graph->V = V;

    // Create an array of adjacency lists.  Size of array will be V
    graph->array = (struct AdjList*) malloc(V * sizeof(struct AdjList));

     // Initialize each adjacency list as empty by making head as NULL
    for (int i = 0; i < V; ++i)
        graph->array[i].head = NULL;

    return graph;
}

// Adds an edge to an undirected graph
void addEdge(struct Graph* graph, int src, int dest, int weight)
{
    // Add an edge from src to dest.  A new node is added to the adjacency
    // list of src.  The node is added at the begining
    struct AdjListNode* newNode = newAdjListNode(dest, weight);
    newNode->next = graph->array[src].head;
    graph->array[src].head = newNode;

    // Since graph is undirected, add an edge from dest to src also
    newNode = newAdjListNode(src, weight);
    newNode->next = graph->array[dest].head;
    graph->array[dest].head = newNode;
}

// Structure to represent a min heap node
struct MinHeapNode
{
    int  v;
    int key;
};

// Structure to represent a min heap
struct MinHeap
{
    int size;      // Number of heap nodes present currently
    int capacity;  // Capacity of min heap
    int *pos;     // This is needed for decreaseKey()
    struct MinHeapNode **array;
};

// A utility function to create a new Min Heap Node
struct MinHeapNode* newMinHeapNode(int v, int key)
{
    struct MinHeapNode* minHeapNode =
           (struct MinHeapNode*) malloc(sizeof(struct MinHeapNode));
    minHeapNode->v = v;
    minHeapNode->key = key;
    return minHeapNode;
}

// A utilit function to create a Min Heap
struct MinHeap* createMinHeap(int capacity)
{
    struct MinHeap* minHeap =
         (struct MinHeap*) malloc(sizeof(struct MinHeap));
    minHeap->pos = (int *)malloc(capacity * sizeof(int));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array =
         (struct MinHeapNode**) malloc(capacity * sizeof(struct MinHeapNode*)
    return minHeap;
}

// A utility function to swap two nodes of min heap. Needed for min heapify
void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b)
{
    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}

// A standard function to heapify at given idx
// This function also updates position of nodes when they are swapped.
// Position is needed for decreaseKey()
void minHeapify(struct MinHeap* minHeap, int idx)
{
    int smallest, left, right;
    smallest = idx;
    left = 2 * idx + 1;
    right = 2 * idx + 2;
    if (left < minHeap->size &&
        minHeap->array[left]->key < minHeap->array[smallest]->key )
      smallest = left;

    if (right < minHeap->size &&
        minHeap->array[right]->key < minHeap->array[smallest]->key )
      smallest = right;

    if (smallest != idx)
    {
        // The nodes to be swapped in min heap
        MinHeapNode *smallestNode = minHeap->array[smallest];
        MinHeapNode *idxNode = minHeap->array[idx];

        // Swap positions
        minHeap->pos[smallestNode->v] = idx;
        minHeap->pos[idxNode->v] = smallest;

        // Swap nodes
        swapMinHeapNode(&minHeap->array[smallest], &minHeap->array[idx]);

        minHeapify(minHeap, smallest);
    }
}

// A utility function to check if the given minHeap is ampty or not
int isEmpty(struct MinHeap* minHeap)
{
    return minHeap->size == 0;
}

// Standard function to extract minimum node from heap
struct MinHeapNode* extractMin(struct MinHeap* minHeap)
{
    if (isEmpty(minHeap))
        return NULL;

    // Store the root node
    struct MinHeapNode* root = minHeap->array[0];

    // Replace root node with last node
    struct MinHeapNode* lastNode = minHeap->array[minHeap->size - 1];
    minHeap->array[0] = lastNode;

    // Update position of last node
    minHeap->pos[root->v] = minHeap->size-1;
    minHeap->pos[lastNode->v] = 0;

    // Reduce heap size and heapify root
    --minHeap->size;
    minHeapify(minHeap, 0);

    return root;
```

```
}

// Function to decreasy key value of a given vertex v. This function
// uses pos[] of min heap to get the current index of node in min heap
void decreaseKey(struct MinHeap* minHeap, int v, int key)
{
    // Get the index of v in  heap array
    int i = minHeap->pos[v];

    // Get the node and update its key value
    minHeap->array[i]->key = key;

    // Travel up while the complete tree is not hepified.
    // This is a O(Logn) loop
    while (i && minHeap->array[i]->key < minHeap->array[(i - 1) / 2]->key)
    {
        // Swap this node with its parent
        minHeap->pos[minHeap->array[i]->v] = (i-1)/2;
        minHeap->pos[minHeap->array[(i-1)/2]->v] = i;
        swapMinHeapNode(&minHeap->array[i],  &minHeap->array[(i - 1) / 2]);

        // move to parent index
        i = (i - 1) / 2;
    }
}

// A utility function to check if a given vertex
// 'v' is in min heap or not
bool isInMinHeap(struct MinHeap *minHeap, int v)
{
   if (minHeap->pos[v] < minHeap->size)
     return true;
   return false;
}

// A utility function used to print the constructed MST
void printArr(int arr[], int n)
{
    for (int i = 1; i < n; ++i)
        printf("%d - %d\n", arr[i], i);
}

// The main function that constructs Minimum Spanning Tree (MST)
// using Prim's algorithm
void PrimMST(struct Graph* graph)
{
    int V = graph->V;// Get the number of vertices in graph
    int parent[V];   // Array to store constructed MST
    int key[V];      // Key values used to pick minimum weight edge in cut

    // minHeap represents set E
    struct MinHeap* minHeap = createMinHeap(V);
    // Initialize min heap with all vertices. Key value of
    // all vertices (except 0th vertex) is initially infinite
    for (int v = 1; v < V; ++v)
    {
        parent[v] = -1;
        key[v] = INT_MAX;
        minHeap->array[v] = newMinHeapNode(v, key[v]);
        minHeap->pos[v] = v;
    }

    // Make key value of 0th vertex as 0 so that it
    // is extracted first
    key[0] = 0;
    minHeap->array[0] = newMinHeapNode(0, key[0]);
    minHeap->pos[0]   = 0;

    // Initially size of min heap is equal to V
    minHeap->size = V;

    // In the followin loop, min heap contains all nodes
    // not yet added to MST.
    while (!isEmpty(minHeap))
    {
        // Extract the vertex with minimum key value
        struct MinHeapNode* minHeapNode = extractMin(minHeap);
        int u = minHeapNode->v; // Store the extracted vertex number

        // Traverse through all adjacent vertices of u (the extracted
        // vertex) and update their key values
        struct AdjListNode* pCrawl = graph->array[u].head;
        while (pCrawl != NULL)
        {
            int v = pCrawl->dest;

            // If v is not yet included in MST and weight of u-v is
            // less than key value of v, then update key value and
            // parent of v
            if (isInMinHeap(minHeap, v) && pCrawl->weight < key[v])
            {
                key[v] = pCrawl->weight;
                parent[v] = u;
                decreaseKey(minHeap, v, key[v]);
            }
            pCrawl = pCrawl->next;
        }
    }

    // print edges of MST
    printArr(parent, V);
}

// Driver program to test above functions
int main()
```

```
{
    // Let us create the graph given in above fugure
    int V = 9;
    struct Graph* graph = createGraph(V);
    addEdge(graph, 0, 1, 4);
    addEdge(graph, 0, 7, 8);
    addEdge(graph, 1, 2, 8);
    addEdge(graph, 1, 7, 11);
    addEdge(graph, 2, 3, 7);
    addEdge(graph, 2, 8, 2);
    addEdge(graph, 2, 5, 4);
    addEdge(graph, 3, 4, 9);
    addEdge(graph, 3, 5, 14);
    addEdge(graph, 4, 5, 10);
    addEdge(graph, 5, 6, 2);
    addEdge(graph, 6, 7, 1);
    addEdge(graph, 6, 8, 6);
    addEdge(graph, 7, 8, 7);

    PrimMST(graph);

    return 0;
}
```

Output:

```
0 - 1
5 - 2
2 - 3
3 - 4
6 - 5
7 - 6
0 - 7
2 - 8
```

Time Complexity: The time complexity of the above code/algorithm looks O(V^2) as there are two nested while loops. If we take a closer look, we can observe that the statements in inner loop are executed O(V+E) times (similar to BFS). The inner loop has decreaseKey() operation which takes O(LogV) time. So overall time complexity is O(E+V)*O(LogV) which is O((E+V)*LogV) = O(ELogV) (For a connected graph, V = O(E))

References:

Introduction to Algorithms by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L.

http://en.wikipedia.org/wiki/Prim's_algorithm

This article is compiled by Aashish Barnwal and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Topics:

- Assign directions to edges so that the directed graph remains acyclic
- K Centers Problem | Set 1 (Greedy Approximate Algorithm)
- Find the minimum cost to reach destination using a train
- Applications of Breadth First Traversal
- Optimal read list for given number of days
- Print all paths from a given source to a destination
- Minimize Cash Flow among a given set of friends who have borrowed money from each other
- Boggle (Find all possible words in a board of characters)

Tags: Graph, Greedy Algorithm

Tweet 3 g+1 0

Writing code in comment? Please use **ideone.com** and share the link here.

27 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

Harsh Singh • 12 days ago
This is a smaller and better implementation in C++ using STL..
http://ideone.com/rQBxov
• Reply • Share ›

Goku • a month ago
http://ideone.com/droqa6 using STL
• Reply • Share ›

Itachi Uchiha • a month ago
Can anyone provide the cpp implementation?
• Reply • Share ›

Aditya Goel • 3 months ago
Wish I had patience to read the whole code, but i guess knowing the algorithm will be enough. Nobody will expect you to implement this program in interview in C.
• Reply • Share ›

jbird • 4 months ago
I try using the implementation to see how close two names are together but when using PrimMST on my graph, it says there is a Segmentation fault in the isInMinHeap function. Any guesses as to what may cause that error?
• Reply • Share ›

Reply • Share ›

Kshitij Bhutani • 7 months ago

#include<bits stdc++.h="">

using namespace std;

#define s(n) scanf("%d",&n)

#define p(n) printf("%d\n",n)

#define forall(i,a,b) for(int i=a;i<b;i++) #define="" first="" fr="" #define="" second="" sd="" #define="" pb="" push_back="" #define="" make_pair="" mp="" #define="" it(t)="" t="" ::="" iterator="" it="" #define="" rep(it,s)="" for(it="s.begin();it!=s.end();it++)" #define="" ll="" long="" long="" int="" #define="" p(p)="" cout<<p<<endl;="" #define="" f(g1,g2)="" g1##g2="" typedef="" vector<int=""> vi;

typedef queue<int> qi;

typedef vector<string> vs;

typedef pair<int,int> pii;

---

see more

Reply • Share ›

Guest • 7 months ago

void printArr(int arr[], int n)
{
for (int i = 1; i < n; ++i)
printf("%d - %d\n", arr[i], i);
}
i should start from zero or not??

Reply • Share ›

iyer → Guest • 7 months ago

no. because parent of i=0 is -1 and we dont print that

Reply • Share ›

Rakesh Agarwal • 7 months ago

are we using parent anywhere in this code ?

Reply • Share ›

Rakesh Agarwal → Rakesh Agarwal • 7 months ago

ok, got it !!

Reply • Share ›

• Reply • Share ›

Kim Jong-il • 8 months ago

I think its hell difficult implementation, it can be implemented really very easily.

• Reply • Share ›

helper Kim Jong-il • 7 months ago

no, i feel it is a systematic and managable implementation if you go with c. java has priority queue built in, so it saves much work by using:
Queue<edge> x= new PriorityQueue<edge>();

• Reply • Share ›

lilly • 10 months ago

could you explain why we are using "decreaseKey(minHeap, v, key[v])" method?
thank you

• Reply • Share ›

samthebest lilly • 10 months ago

we are updating the key in the heap ...as if we find a another smaller edge through which the already visted vertices MST can be connected

• Reply • Share ›

Stormey • a year ago

Shouldn't it be: "Key is updated if the a vertex is in Min Heap and previous key value is greater than the weight of edge from X to the adjacent"?
Since you DO want to update vertex's key ONLY if they are in the MinHeap...

• Reply • Share ›

Asap • 2 years ago

I think we dont need 2 key arrays.
Correct me if i am wrong.

• Reply • Share ›

Abhishek Asap • 2 years ago

I agree. We don't need two 2 key arrays.
Inside the PrimMST function call :

```
while (pCrawl != NULL)
{
..
key[v] = pCrawl->weight;
.....
decreaseKey(minHeap, v, key[v]);
}
}
```

We update the key values twice.

key[v] =
as well as inside the decreaseKey() function call
minHeap->array[i]->key = key;

At any point of time, they will have equal values.

• Reply • Share ›

MK Abhishek • 10 months ago

But removing key[] giving a different answer!!! Why?

• Reply • Share ›

Kumar Vikram • 2 years ago

Another implementation using Adjancency list in c++.

```
[sourcecode language="C++"]
#include<iostream>
#include <list>
#include <limits.h>

using namespace std;

class AdjListNode
{
int v;
int weight;
public:
AdjListNode(int _v, int _w) { v = _v; weight = _w;}
int getV() { return v; }
int getWeight() { return weight; }
};
```

see more

• Reply • Share ›

jaskaran1 • 2 years ago

Coded up a shorter implementation using STL and found another MST
0-1
1-2
2-8
2-5
5-6
6-7

2-3
3-4

```
/* Paste your code here (You may delete these lines if not writing code) */
```

4 • Reply • Share ›

Sandeep Jain • 2 years ago
Parent value -1 indicates root of MST. There should be only one -1 for a connected undirected graph.. If the graph is disconnected, then output is a forest (more than one -1 values)
• Reply • Share ›

Somnath Dutta • 2 years ago
I am using the code to build a mst having around 300 vertices but some of the values of parent array has value -1, iam confused , can anybody suggest me something?
• Reply • Share ›

Anurag Singh Somnath Dutta • 8 months ago
parent array is -1 after running the program on the graph ??, probably because those nodes are not reachable (except the one which is actually root). It may happen when your graph is not connected (i.e. it's a forest)
• Reply • Share ›

golu • 2 years ago
i tried so many times to implement it, but failed

it really needs courage to write such a program
2 • Reply • Share ›

Pushkar golu • 2 years ago
it needs programming skills :)
3 • Reply • Share ›

helper Pushkar • 7 months ago
it needs time, you give me a free day and i will write this whole code by myself in c... anyways kudos to the author :)
• Reply • Share ›

helper helper • 6 months ago
as promised, i have written the code myself, and it took me just 1-2 hours.

http://ideone.com/8MU0ln

the complexity of my code is
E lg(E)=E lg(V^2)=2E lg (V)=E lg(V). same as optimal. :)

• Reply • Share ›

Subscribe Add Disqus to your site Privacy

-
-
-
-
  - Interview Experiences
  - Advanced Data Structures
  - Dynamic Programming
  - Greedy Algorithms
  - Backtracking
  - Pattern Searching
  - Divide & Conquer
  - Mathematical Algorithms
  - Recursion
  - Geometric Algorithms
-

- ## Popular Posts

  - All permutations of a given string
  - Memory Layout of C Programs
  - Understanding "extern" keyword in C
  - Median of two sorted arrays
  - Tree traversal without recursion and without stack!
  - Structure Member Alignment, Padding and Data Packing
  - Intersection point of two Linked Lists
  - Lowest Common Ancestor in a BST.

  - Check if a binary tree is BST or not
  - Sorted Linked List to Balanced BST
- Follow @GeeksforGeeks

- ## Recent Comments

  - lt_k

    i need help for coding this function in java...

    Java Programming Language · 1 hour ago
  - Piyush

    What is the purpose of else if (recStack[*i])...

    Detect Cycle in a Directed Graph · 1 hour ago
  - Andy Toh

    My compile-time solution, which agrees with the...

    Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour ago
  - lucy

    because we first fill zero in first col and...

    Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago
  - lucy

    @GeeksforGeeks i don't n know what is this long...

    Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 2 hours ago
  - manish

    Because TAN is not a subsequence of RANT. ANT...

    Given two strings, find if first string is a subsequence of second · 2 hours ago
-