|  |  |  |  |
| --- | --- | --- | --- |
| 2022/2 『자료구조』실습 보고서 | | | |
| 제목 | 11장 실습( O ) 과제( ) | 제출일자 | 2022.  12 .    01 . |
| 학번 | 201911608 | 이름 | 김 지환 |

|  |
| --- |
| 1. 프로그램 11.8 - Kruskal 최소비용 신장 트리 프로그램    #include <stdio.h>  #include <stdlib.h>  #define TRUE 1  #define FALSE 0  #define MAX\_VERTICES 100  #define INF 1000  int parent[MAX\_VERTICES];  int visited[MAX\_VERTICES];  void set\_init(int n) {  for (int i = 0; i < n; i++) parent[i] = -1;  }  int set\_find(int curr) {  if (parent[curr] == -1) return curr;  while (parent[curr] != -1) curr = parent[curr];  return curr;  }  void set\_union(int a, int b) {  int root1 = set\_find(a);  int root2 = set\_find(b);  if (root1 != root2) parent[root1] = root2;  }  struct Edge {  int start, end, weight;  };  typedef struct GraphType {  int n;  struct Edge edges[2 \* MAX\_VERTICES];  } GraphType;  void graph\_init(GraphType\* g) {  g->n = 0;  for (int i = 0; i < 2 \* MAX\_VERTICES; i++) {  g->edges[i].start = 0;  g->edges[i].end = 0;  g->edges[i].weight = INF;  }  }  void insert\_edge(GraphType\* g, int start, int end, int w) {  g->edges[g->n].start = start;  g->edges[g->n].end = end;  g->edges[g->n].weight = w;  if (!visited[start]) {  visited[start] = 1;  g->n++;  }  }  int compare(const void\* a, const void\* b) {  struct Edge\* x = (struct Edge\*)a;  struct Edge\* y = (struct Edge\*)b;  return (x->weight - y->weight);  }    void kruskal(GraphType\* g) {  int edge\_accepted = 0;  int uset, vset;  struct Edge e;  set\_init(g->n);  qsort(g->edges, g->n, sizeof(struct Edge), compare);  //for (int i = 0; i < g->n; i++) printf("%d %d %d \n", g->edges[i].start, g->edges[i].end, g->edges[i].weight);  printf("크루스칼 최소 신장 트리 알고리즘 \n");  int i = 0;  while (edge\_accepted < (g->n-1)) {  e = g->edges[i];  uset = set\_find(e.start);  vset = set\_find(e.end);  //printf("\n error - %d, %d \n", uset, vset);  if (uset != vset) {  printf("간선 (%d,%d) %d 선택\n", e.start, e.end, e.weight);  edge\_accepted++;  set\_union(uset, vset);  }  i++;  }  }  int main(void) {  GraphType\* g;  g = (GraphType\*)malloc(sizeof(GraphType));  graph\_init(g);  insert\_edge(g, 0, 1, 29);  insert\_edge(g, 1, 2, 16);  insert\_edge(g, 2, 3, 12);  insert\_edge(g, 3, 4, 22);  insert\_edge(g, 4, 5, 27);  insert\_edge(g, 5, 0, 10);  insert\_edge(g, 6, 1, 15);  insert\_edge(g, 6, 3, 18);  insert\_edge(g, 6, 4, 25);  kruskal(g);  free(g);  return 0;  }  실행결과    -> insert를 할 때 g->n++ 부분이 노드 개수만큼 되지 않고 간선 개수만큼 증가되어서  방문 처리를 이용해 노드 개수만큼 g->n++을 이용해 무한루프를 해결하였다.  추가적으로 weight을 기준으로 오름차순 정렬된 간선에서 방문하지 않은 노드를 -1로 방문한 노드를 start 노드에서 end 노드로 합집합 처리했다.  이후 start노드를 방문하지 않은 노드를 찾아 합집합 처리를 노드의개수 -1 번 만큼 반복한다.  그 이유는 0~6번 까지 노드가 있으면 0~5번까지 노드를 합하고 남은 6번 노드는 0~5번 노드 중 하나와 연결되어 있기 때문이다. |
| 2. Quiz (408쪽) - 그래프를 적용한 Kruskal 최소 비용신장 트리 형성 과정  01. 아래 그래프에 Kruskal 알고리즘을 이용해 최소 비용 신장 트리를 만드는 과정을 보여라.  {0, 1} = 10 {0, 3} = 6 {0, 7} = 1  {1, 5} = 2 {1, 2} = 4 {2, 3} = 11  {2, 5} = 7 {4, 5} = 5 {4, 7} = 8  {5, 6| = 9 {6, 7} = 12  -> 해당 그래프를 간선으로 나타내면 아래와 같다. 정점은 0번~6번, 간선은 11개이다.  #include <stdio.h>  #include <stdlib.h>  #define TRUE 1  #define FALSE 0  #define MAX\_VERTICES 100  #define INF 1000  int parent[MAX\_VERTICES];  int visited[MAX\_VERTICES];  void set\_init(int n) {  for (int i = 0; i < n; i++) parent[i] = -1;  }  int set\_find(int curr) {  if (parent[curr] == -1) return curr;  while (parent[curr] != -1) curr = parent[curr];  return curr;  }  void set\_union(int a, int b) {  int root1 = set\_find(a);  int root2 = set\_find(b);  if (root1 != root2) parent[root1] = root2;  }  struct Edge {  int start, end, weight;  };  typedef struct GraphType {  int n;  struct Edge edges[2 \* MAX\_VERTICES];  } GraphType;  void graph\_init(GraphType\* g) {  g->n = 0;  for (int i = 0; i < 2 \* MAX\_VERTICES; i++) {  g->edges[i].start = 0;  g->edges[i].end = 0;  g->edges[i].weight = INF;  }  }  void insert\_edge(GraphType\* g, int start, int end, int w) {  g->edges[g->n].start = start;  g->edges[g->n].end = end;  g->edges[g->n].weight = w;  if (!visited[start]) {  visited[start] = 1;  g->n++;  }  else if (!visited[end]) {  visited[end] = 1;  g->n++;  }  }  int compare(const void\* a, const void\* b) {  struct Edge\* x = (struct Edge\*)a;  struct Edge\* y = (struct Edge\*)b;  return (x->weight - y->weight);  }    void kruskal(GraphType\* g) {  int edge\_accepted = 0;  int uset, vset;  struct Edge e;  set\_init(g->n);  qsort(g->edges, g->n, sizeof(struct Edge), compare);  int dis = 0;  //for (int i = 0; i < g->n; i++) printf("%d %d %d \n", g->edges[i].start, g->edges[i].end, g->edges[i].weight);  printf("크루스칼 최소 신장 트리 알고리즘 \n");  int i = 0;  while (edge\_accepted < (g->n-1)) {  e = g->edges[i];  uset = set\_find(e.start);  vset = set\_find(e.end);  //printf("\n error - %d, %d \n", uset, vset);  if (uset != vset) {  printf("간선 (%d,%d) %d 선택\n", e.start, e.end, e.weight);  dis += e.weight;  edge\_accepted++;  set\_union(uset, vset);  }  i++;  }  printf("총 거리는 %d이다.", dis);  }  int main(void) {  GraphType\* g;  g = (GraphType\*)malloc(sizeof(GraphType));  graph\_init(g);  insert\_edge(g, 0, 1, 10);  insert\_edge(g, 0, 3, 6);  insert\_edge(g, 0, 7, 1);  insert\_edge(g, 1, 2, 4);  insert\_edge(g, 1, 5, 2);  insert\_edge(g, 2, 3, 11);  insert\_edge(g, 2, 5, 7);  insert\_edge(g, 4, 5, 5);  insert\_edge(g, 4, 7, 8);  insert\_edge(g, 5, 6, 9);  insert\_edge(g, 6, 7, 12);  kruskal(g);  free(g);  return 0;  }  ->    해당 문제도 방문처리 이후 모든 간선을 최소비용으로 연결했다. |
| 3. 프로그램 11.9 - Prim 최소 비용 신장 트리 프로그램  #include <stdio.h>  #include <stdlib.h>  #define TRUE 1  #define FALSE 0  #define MAX\_VERTICES 100  #define INF 1000L  typedef struct GraphType {  int n;  int weight[MAX\_VERTICES][MAX\_VERTICES];  } GraphType;  int selected[MAX\_VERTICES];  int distance[MAX\_VERTICES];  int get\_min\_vertex(int n) {  int v, i;  for (i = 0; i < n; i++)  if (!selected[i]) {  v = i;  break;  }  for (i = 0; i < n; i++)  if (!selected[i] && (distance[i] < distance[v])) v = i;  return v;  }  void prim(GraphType\* g, int s) {  int i, u, v;  for (u = 0; u < g->n; u++) distance[u] = INF;  distance[s] = 0;  for (i = 0; i < g->n; i++) {  u = get\_min\_vertex(g->n);  selected[u] = TRUE;  if (distance[u] == INF) return;  printf("정점 %d 추가\n", u);  for (v = 0; v < g->n; v++)  if (g->weight[u][v] != INF)  if (!selected[v] && g->weight[u][v] < distance[v])  distance[v] = g->weight[u][v];  }  }  int main(void) {  GraphType g = { 7,  {{ 0, 29, INF, INF, INF, 10, INF },  { 29, 0, 16, INF, INF, INF, 15 },  { INF, 16, 0, 12, INF, INF, INF },  { INF, INF, 12, 0, 22, INF, 18 },  { INF, INF, INF, 22, 0, 27, 25 },  { 10, INF, INF, INF, 27, 0, INF },  { INF, 15, INF, 18, 25, INF, 0 } }  };  prim(&g, 0);  return 0;  }  실행결과    -> prim 최소 신장트리 알고리즘은 해당 정점 중 방문하지 않은 가장 작은 정점을 찾아서 모든 정점을 잇는 최단거리를 찾는 알고리즘이다.  1. 0번 정점부터 시작해서 방문하지 않은 정점 중에서 가장 비용이 적은 5번 노드를 선택  2. 5번 정점에서 방문하지 않은 정점 중에서 가장 비용이 적은 4번 노드를 선택  3. 위의 과정을 반복하면 0->5->4->3->2->1->6이 선택된다.    -> 프림과 크루스칼 비교  프림은 우선으로 인접해있는 노드들 간 간선 연결을 하고 연결되지 않는 노드 중에서 인접해있는 노드 중 가장 비용이 적은 간선을 채택한다.  크루스칼은 최적의 거리만 찾기 때문에 그림과 같이 비용이 하나의 노드를 기준으로 방문하지 않은 가장 비용이 적은 노드간 간선으로 연결되어 있다. |
| 4. Quiz (414쪽) - 그래프 적용 Prim 최소 비용 신장 트리 형성 과정  #include <stdio.h>  #include <stdlib.h>  #define TRUE 1  #define FALSE 0  #define MAX\_VERTICES 100  #define INF 1000L  typedef struct GraphType {  int n;  int weight[MAX\_VERTICES][MAX\_VERTICES];  } GraphType;  int selected[MAX\_VERTICES];  int distance[MAX\_VERTICES];  int get\_min\_vertex(int n) {  int v, i;  for (i = 0; i < n; i++)  if (!selected[i]) {  v = i;  break;  }  for (i = 0; i < n; i++)  if (!selected[i] && (distance[i] < distance[v])) v = i;  return v;  }  void prim(GraphType\* g, int s) {  int i, u, v;  for (u = 0; u < g->n; u++) distance[u] = INF;  distance[s] = 0;  for (i = 0; i < g->n; i++) {  u = get\_min\_vertex(g->n);  selected[u] = TRUE;  if (distance[u] == INF) return;  printf("정점 %d 방문!\n", u);  for (v = 0; v < g->n; v++)  if (g->weight[u][v] != INF)  if (!selected[v] && g->weight[u][v] < distance[v]) {  if(distance[v]!=INF)  printf("정점 %d에서 정점 %d의 거리 %d 수정\n", u, v, g->weight[u][v]);  else  printf("정점 %d에서 정점 %d의 거리 %d 추가\n", u, v, g->weight[u][v]);  distance[v] = g->weight[u][v];  }  }  }  int main(void) {  int sum = 0;  GraphType g = { 8,  {{ 0, 10, INF, 6, INF, INF, INF, 1 },  { 10, 0, 4, INF, INF, 2, INF, INF },  { INF, 4, 0, 11, INF, 7, INF, INF },  { 6, INF, 11, 0, INF, INF, INF, 3 },  { INF, INF, INF, INF, 0, 5, INF, 8 },  { INF, 2, INF, INF, 5, 0, 9, INF },  { INF, INF, INF, INF, INF, 9, 0, 12 },  { 1, INF, INF, 3, 8, INF, INF, 0 }}  };  prim(&g, 0);  for (int i = 0; i < g.n;i++) sum += distance[i];  printf("최단 경로 거리 : %d\n", sum);  return 0;  }  실행결과 |