|  |  |  |  |
| --- | --- | --- | --- |
| 2022/2 『자료구조』과제 보고서 | | | |
| 제목 | 11장 실습( ) 과제( O ) | 제출일자 | 2022.  12 .    07 . |
| 학번 | 201911608 | 이름 | 김 지환 |

|  |
| --- |
| 02. 아래의 네트워크에 대하여 KrusKal의 MST 알고리즘을 이용해서 최소비용 신장 트리가 구성되는 과정을 보여라.    #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 (%c,%c) %d 선택\n", e.start+'A', e.end+'A', 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, 7);  insert\_edge(g, 1, 0, 7);  insert\_edge(g, 0, 3, 5);  insert\_edge(g, 3, 0, 5);  insert\_edge(g, 1, 2, 8);  insert\_edge(g, 2, 1, 8);  insert\_edge(g, 1, 3, 9);  insert\_edge(g, 3, 1, 9);  insert\_edge(g, 1, 4, 7);  insert\_edge(g, 4, 1, 7);  insert\_edge(g, 2, 4, 5);  insert\_edge(g, 4, 2, 5);  insert\_edge(g, 3, 4, 15);  insert\_edge(g, 4, 3, 15);  insert\_edge(g, 3, 5, 6);  insert\_edge(g, 5, 3, 6);  insert\_edge(g, 4, 5, 8);  insert\_edge(g, 5, 4, 8);  insert\_edge(g, 4, 6, 9);  insert\_edge(g, 6, 4, 9);  insert\_edge(g, 5, 6, 11);  insert\_edge(g, 6, 5, 11);  kruskal(g);  free(g);  return 0;  }  실행결과      -> 모든 경로 탐색 완료 |
| 04. Prim의 함수에서 distance[]와 selected[]의 값을 출력하는 문장을 삽입하여 출력하여 보고 이들의 의미를 설명하라.  #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);  printf("\*\*\*distance\*\*\*\n");  for (int i = 0; i < g.n; i++)  printf("[%d]", distance[i]);  printf("\n\n\*\*\*selected\*\*\*\n");  for (int i = 0; i < g.n; i++)  printf("[%d]", selected[i]);  printf("\n");  return 0;  }  실행결과    -> distance 는 해당 index 번호의 정점의 최소 거리값을 저장해 보다 짧은 거리가 발견되면 업데이트 하기 위한 용도이다.  -> selected는 방문 했는지 체크하는 용도이다. |
| 06. 5번의 그래프에 대하여 Floyd 알고리즘을 적용하고 배열 A의 내용을 매 단계마다 출력하라.    그래프를 배열로 표시하면 그림과 같다.  중간 노드 = 0,  20+50( i = 3, j = 1), 20+45(i=3, j=2), 20+10 (i= 3, j = 3) 을 할 수 있다.  더한 값보다 작은 경우는 graph\_31, graph\_32 이므로 70과 65를 삽입한다.    중간 노드 = 1,  70+10(i=3, j=2), 70+15(i=3, j=3), 20+10(i=4, j=2), 20+15(i=4, j=3) 을 할 수 있다.  그 중 조건을 만족하는 경우는 (4,2),(4,3) 이다.    중간 노드 = 2,  조건을 만족하는 (0, 4), (1, 4), (3, 4) 삽입    위와 같은 과정을 반복 |
| 08. 최단 경로 함수를 최단 경로의 길이 뿐만 아니라 그 경로까지 출력할 수 있도록 수정하라.  #include <stdio.h>  #include <stdlib.h>  #include <limits.h>  #define TRUE 1  #define FALSE 0  #define MAX\_VERTICES 100  #define INF 1000000  typedef struct GraphType {  int n;  int weight[MAX\_VERTICES][MAX\_VERTICES];  } GraphType;  int cnt;  int distance[MAX\_VERTICES];  int found[MAX\_VERTICES];  int path[MAX\_VERTICES];  int choose(int distance[], int n, int found[]) {  int i, min, minpos;  min = INT\_MAX;  minpos = -1;  for (i = 0; i < n; i++)  if (distance[i] < min && !found[i]) {  min = distance[i];  minpos = i;  }  return minpos;  }  void print\_status(GraphType\* g) {  static int step = 1;  printf("STEP %d: ", step++);  printf("distance: ");  for (int i = 0; i < g->n; i++) {  if (distance[i] == INF) printf(" \* ");  else printf("%2d ", distance[i]);  }  printf("\n");  printf(" found: ");  for (int i = 0; i < g->n; i++) printf("%2d ", found[i]);  printf("\n\n");  }  void shortest\_path(GraphType\* g, int start) {  int i, u, w;  for (i = 0; i < g->n; i++) {  distance[i] = g->weight[start][i];  found[i] = FALSE;  }  found[start] = TRUE;  path[cnt++] = start;  for (i = 0; i < g->n - 1; i++) {  print\_status(g);  u = choose(distance, g->n, found);  found[u] = TRUE;  path[cnt++] = u;  for (w = 0; w < g->n; w++)  if (!found[w] && (distance[u] + g->weight[u][w] < distance[w]))  distance[w] = distance[u] + g->weight[u][w];  }  }  int main(void) {  GraphType g = { 7,  {{ 0, 7, INF, INF, 3, 10, INF },  { 7, 0, 4, 10, 2, 6, INF },  { INF, 4, 0, 2, INF, INF, INF },  { INF, 10, 2, 0, 11, 9, 4 },  { 3, 2, INF, 11, 0, INF, 5 },  { 10, 6, INF, 9, INF, 0, INF },  { INF, INF, INF, 4, 5, INF, 0 } }  };  shortest\_path(&g, 0);  printf("\*\*\* 다익스트라 최단경로 \*\*\*\n");  int dis = 0;  for (int i = 0; i < g.n; i++) {  dis += distance[i];  printf("[%d] -> ", path[i]);  }  printf("\n\n최단경로 총 길이 : %d\n", dis);  return 0;  }  실행결과 |
| 10. 다음의 그래프에 대하여 위상 정렬을 적용하고 그 결과를 구하라.    #include <stdio.h>  #include <stdlib.h>  #define TRUE 1  #define FALSE 0  #define MAX\_VERTICES 50  typedef struct GraphNode {  int vertex;  struct GraphNode\* link;  } GraphNode;  typedef struct GraphType {  int n;  GraphNode\* adj\_list[MAX\_VERTICES];  } GraphType;  void graph\_init(GraphType\* g) {  int v;  g->n = 0;  for (v = 0; v < MAX\_VERTICES; v++)  g->adj\_list[v] = NULL;  }  void insert\_vertex(GraphType\* g, int v) {  if (((g->n) + 1) > MAX\_VERTICES) {  fprintf(stderr, "그래프: 정점의 개수 초과");  return;  }  g->n++;  }  void insert\_edge(GraphType\* g, int u, int v) {  GraphNode\* node;  if (u >= g->n || v >= g->n) {  fprintf(stderr, "그래프: 정점 번호 오류");  return;  }  node = (GraphNode\*)malloc(sizeof(GraphNode));  node->vertex = v;  node->link = g->adj\_list[u];  g->adj\_list[u] = node;  }  #define MAX\_STACK\_SIZE 100  typedef int element;  typedef struct {  element stack[MAX\_STACK\_SIZE];  int top;  } StackType;  void init(StackType\* s) {  s->top = -1;  }  int is\_empty(StackType\* s) {  return (s->top == -1);  }  int is\_full(StackType\* s) {  return (s->top == (MAX\_STACK\_SIZE - 1));  }  void push(StackType\* s, element item) {  if (is\_full(s)) {  fprintf(stderr, "스택 포화 에러\n");  return;  }  else s->stack[++(s->top)] = item;  }  element pop(StackType\* s) {  if (is\_empty(s)) {  fprintf(stderr, "스택 공백 에러\n");  exit(1);  }  else return s->stack[(s->top)--];  }  int topo\_sort(GraphType\* g) {  int i;  StackType s;  GraphNode\* node;  int\* in\_degree = (int\*)malloc(g->n \* sizeof(int));  for (i = 0; i < g->n; i++)  in\_degree[i] = 0;  for (i = 0; i < g->n; i++) {  GraphNode\* node = g->adj\_list[i];  while (node != NULL) {  in\_degree[node->vertex]++;  node = node->link;  }  }  init(&s);  for (i = 0; i < g->n; i++) {  if (in\_degree[i] == 0) push(&s, i);  }  while (!is\_empty(&s)) {  int w;  w = pop(&s);  printf("(CS%d) -> ", w+1);  node = g->adj\_list[w];  while (node != NULL) {  int u = node->vertex;  in\_degree[u]--;  if (in\_degree[u] == 0) push(&s, u);  node = node->link;  }  }  free(in\_degree);  printf("\n");  return (i == g->n);  }  int main(void) {  GraphType g;  graph\_init(&g);  for(int i = 0; i<8; i++)  insert\_vertex(&g, i);  insert\_edge(&g, 0, 1);  insert\_edge(&g, 1, 2);  insert\_edge(&g, 1, 3);  insert\_edge(&g, 1, 4);  insert\_edge(&g, 2, 7);  insert\_edge(&g, 3, 6);  insert\_edge(&g, 4, 5);  insert\_edge(&g, 4, 6);  insert\_edge(&g, 5, 6);  insert\_edge(&g, 6, 7);  topo\_sort(&g);  return 0;  }  실행결과 |