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

|  |
| --- |
| 1. 프로그램 11.10 - 최단 경로 Dijkstra 프로그램  #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 distance[MAX\_VERTICES];  int found[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;  for (i = 0; i < g->n - 1; i++) {  print\_status(g);  u = choose(distance, g->n, found);  found[u] = TRUE;  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);  return 0;  }  실행결과    ->    0 번 정점에서 탐색 최소값인 4번 정점 탐색 최소값 1번 정점 탐색    최소값 6번 정점 탐색 최소값 2번 정점 탐색  -> 이후 탐색해도 최선의 값이 없음  0-4-1-2-3-6-5 가 최단 경로이다. |
| 2. 11.11 - Floyd 최단 경로 프로그램  #include <stdio.h>  #include <stdlib.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 A[MAX\_VERTICES][MAX\_VERTICES];  void printA(GraphType\* g) {  int i, j;  printf("===============================\n");  for (i = 0; i < g->n; i++) {  for (j = 0; j < g->n; j++) {  if (A[i][j] == INF)  printf(" \* ");  else printf("%3d ", A[i][j]);  }  printf("\n");  }  printf("===============================\n");  }  void floyd(GraphType\* g) {  int i, j, k;  for (i = 0; i < g->n; i++)  for (j = 0; j < g->n; j++)  A[i][j] = g->weight[i][j];  printA(g);  for (k = 0; k < g->n; k++) {  for (i = 0; i < g->n; i++)  for (j = 0; j < g->n; j++)  if (A[i][k] + A[k][j] < A[i][j])  A[i][j] = A[i][k] + A[k][j];  printA(g);  }  }  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 } }  };  floyd(&g);  return 0;  }  실행결과    ->  각 정점간의 최소 거리를 찾는다. |
| 3. 프로그램 11.13 - 위상 정렬 프로그램  #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("정점 %d ->", w);  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);  insert\_vertex(&g, 0);  insert\_vertex(&g, 1);  insert\_vertex(&g, 2);  insert\_vertex(&g, 3);  insert\_vertex(&g, 4);  insert\_vertex(&g, 5);  insert\_edge(&g, 0, 2);  insert\_edge(&g, 0, 3);  insert\_edge(&g, 1, 3);  insert\_edge(&g, 1, 4);  insert\_edge(&g, 2, 3);  insert\_edge(&g, 2, 5);  insert\_edge(&g, 3, 5);  insert\_edge(&g, 4, 5);  topo\_sort(&g);  return 0;  }  실행결과    -> 진입차수 0인 것 = 0, 1 스택 = {0, 1}  1. 1번 정점 방문 후 4번, 3번 정점의 진입차수 - 1, 4번 정점의 진입차수 = 0  -> 스택 = {0, 4}  2. 4번 정점 방문 후 5번 정점 진입차수 - 1  -> 스택 = {0}  3. 0번 정점 방문 후 3번, 4번 정점의 진입차수 - 1, 2번 정점의 진입차수 = 0  -> 스택 = {2}  4. 2번 정점 방문 후 5번,4번 진입 차수 - 1, 3번 정점 진입차수 = 0  -> 스택 = {3}  5. 3번 정점 방문 후 5번 정점 진입 차수 - 1, 5번 정점 진입차수 = 0  -> 스택 = {5}  스택 출력 순서 = 위상정렬 = 1, 4, 0, 2, 3, 5  // 제대로 실행 되었음을 알 수 있다. |