|  |  |  |  |
| --- | --- | --- | --- |
| 2022/2 『자료구조』실습 보고서 | | | |
| 제목 | 10장 실습( O ) 과제( ) | 제출일자 | 2022.  11 .    30 . |
| 학번 | 201911608 | 이름 | 김지환 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. 프로그램 10.1 - 인접 행렬 구성과 그래프 연결 관계 출력(dump)  #include <stdio.h>  #include <stdlib.h>  #define MAX\_VERTICES 50  typedef struct GraphType {  int n;  int adj\_mat[MAX\_VERTICES][MAX\_VERTICES];  } GraphType;  void init(GraphType\* g) {  int r, c;  g->n = 0;  for (r = 0; r < MAX\_VERTICES; r++)  for (c = 0; c < MAX\_VERTICES; c++)  g->adj\_mat[r][c] = 0;  }  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 start, int end) {  if (start >= g->n || end >= g->n) {  fprintf(stderr, "그래프: 정점 번호 오류");  return;  }  g->adj\_mat[start][end] = 1;  g->adj\_mat[end][start] = 1;  }  void print\_adj\_mat(GraphType\* g) {  for (int i = 0; i < g->n; i++) {  for (int j = 0; j < g->n; j++) {  printf("%2d ", g->adj\_mat[i][j]);  }  printf("\n");  }  }  int main() {  GraphType\* g;  g = (GraphType\*)malloc(sizeof(GraphType));  init(g);  for (int i = 0;i < 4;i++)  insert\_vertex(g, i);  insert\_edge(g, 0, 1);  insert\_edge(g, 0, 2);  insert\_edge(g, 0, 3);  insert\_edge(g, 1, 2);  insert\_edge(g, 2, 3);  print\_adj\_mat(g);  free(g);  return 0;  }  출력결과    -> 인접 행렬로 표현하였다.  insert\_edge(g, 0, 1); // 1번 노드와 0번 노드 연결 (0, 1) - (1, 0)  insert\_edge(g, 0, 2); // 2번 노드와 0번 노드 연결 (0, 2) - (2, 0)  insert\_edge(g, 0, 3); // 3번 노드와 0번 노드 연결 (0, 3) - (3, 0)  insert\_edge(g, 1, 2); // 2번 노드와 1번 노드 연결 (1, 2) - (2, 1)  insert\_edge(g, 2, 3); // 3번 노드와 2번 노드 연결 (2, 3) - (3, 2)  0~3번째 index에서 위 값들만 간선으로 연결 되었으니 아래와 같이 구성 된다.   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | | 0 |  | 1 | 1 | 1 | | 1 | 1 |  | 1 |  | | 2 | 1 | 1 |  | 1 | | 3 | 1 |  | 1 |  | |
| 2. 프로그램 10.2 - 인접 리스트 구성과 그래프 연결 관계 출력(dump)  #include <stdio.h>  #include <stdlib.h>  #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 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;  }  void print\_adj\_list(GraphType\* g) {  for (int i = 0; i < g->n; i++) {  GraphNode\* p = g->adj\_list[i];  printf("정점 %d의 인접 리스트 ", i);  while (p != NULL) {  printf("-> %d ", p->vertex);  p = p->link;  }  printf("\n");  }  }  int main() {  GraphType\* g;  g = (GraphType\*)malloc(sizeof(GraphType));  init(g);  for (int i = 0;i < 4;i++)  insert\_vertex(g, i);  insert\_edge(g, 0, 1);  insert\_edge(g, 1, 0);  insert\_edge(g, 0, 2);  insert\_edge(g, 2, 0);  insert\_edge(g, 0, 3);  insert\_edge(g, 3, 0);  insert\_edge(g, 1, 2);  insert\_edge(g, 2, 1);  insert\_edge(g, 2, 3);  insert\_edge(g, 3, 2);  print\_adj\_list(g);  free(g);  return 0;  }  실행결과    -> 인접리스트는 선형 그래프이다.  4개의 리스트를 선언 후 삽입한다.  insert\_edge(g, 0, 1); // g[0] = 1 -> null  insert\_edge(g, 1, 0); // g[1] = 0 -> null  insert\_edge(g, 0, 2); // g[0] = 2 -> 1 -> null  insert\_edge(g, 2, 0); // g[2] = 0 -> null  insert\_edge(g, 0, 3); // g[0] = 3 -> 2-> 1 -> null  insert\_edge(g, 3, 0); // g[3] = 0 -> null  insert\_edge(g, 1, 2); // g[1] = 2 -> 0 -> null  insert\_edge(g, 2, 1); // g[2] = 1 -> 0 -> null  insert\_edge(g, 2, 3); // g[2] = 3 -> 1-> 0 -> null  insert\_edge(g, 3, 2); // g[3] = 2 -> 0 -> null |
| 3. 프로그램 10.3 - 인접 행렬 그래프의 깊이우선탐색 프로그램  #include <stdio.h>  #include <stdlib.h>  #define TRUE 1  #define FALSE 0  #define MAX\_VERTICES 50  typedef struct GraphType {  int n;  int adj\_mat[MAX\_VERTICES][MAX\_VERTICES];  } GraphType;  int visited[MAX\_VERTICES];  void init(GraphType\* g) {  int r, c;  g->n = 0;  for (r = 0; r < MAX\_VERTICES; r++)  for (c = 0; c < MAX\_VERTICES; c++)  g->adj\_mat[r][c] = 0;  }  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 start, int end) {  if (start >= g->n || end >= g->n) {  fprintf(stderr, "그래프: 정점 번호 오류");  return;  }  g->adj\_mat[start][end] = 1;  g->adj\_mat[end][start] = 1;  }  void dfs\_mat(GraphType\* g, int v) {  int w;  visited[v] = TRUE;  printf("정점 %d -> ", v);  for (w = 0; w < g->n; w++)  if (g->adj\_mat[v][w] && !visited[w])  dfs\_mat(g, w);  }  int main(void) {  GraphType\* g;  g = (GraphType\*)malloc(sizeof(GraphType));  init(g);  for (int i = 0; i < 4; i++)  insert\_vertex(g, i);  insert\_edge(g, 0, 1);  insert\_edge(g, 0, 2);  insert\_edge(g, 0, 3);  insert\_edge(g, 1, 2);  insert\_edge(g, 2, 3);  printf("깊이 우선 탐색\n");  dfs\_mat(g, 0);  printf("\n");  free(g);  return 0;  }  실행결과    ->  dfs를 0부터 시작하므로 0의 인접 행렬은 1, 2, 3 이다.  step 1. 0번 노드 방문 처리 후 [0][0] ~ [0][3] 까지 탐색  step 2. 0번 노드인 [0][0]은 이미 탐색 하였으므로 [0][1]을 탐색  step 3. 1번 노드 방문 처리 후 [1][0] ~ [1][3] 까지 탐색  step 4. 0번 노드와 1번 노드인 [1][0], [1][1]은 탐색 하였으므로 [1][2]를 탐색  step 5. 2번 노드 방문 처리 후 [2][0] ~ [2][3] 까지 탐색  step 6. 0번 ~ 2번 노드는 탐색 하였으므로 [2][3] 탐색  step 7. 3번 노드 방문 처리까지 완료. 남은 [2][0]~[2][3]은 이미 방문 처리가 다 되었으므로 종료  step 8. 스텝 5, 3, 1에서 남은 재귀 스택까지 종료  -> 최종적으로 0 -> 1 -> 2 -> 3번 노드 순으로 방문한다. |
| 4. Quiz(391쪽) 01 - 문제에서 제시된 그래프에 프로그램 10.6 (인접 리스트)에 적용  #include <stdio.h>  #include <stdlib.h>  #define MAX\_VERTICES 50  #define TRUE 1  #define FALSE 0  int visited[MAX\_VERTICES];  typedef struct GraphNode {  int vertex;  struct GraphNode\* link;  } GraphNode;  typedef struct GraphType {  int n;  GraphNode\* adj\_list[MAX\_VERTICES];  } GraphType;  typedef struct {  int queue[MAX\_VERTICES];  int front, rear;  }QueueType;  void 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;  }  void print\_adj\_list(GraphType\* g) {  for (int i = 0; i < g->n; i++) {  GraphNode\* p = g->adj\_list[i];  printf("정점 %d의 인접 리스트 ", i);  while (p != NULL) {  printf("-> %d ", p->vertex);  p = p->link;  }  printf("\n");  }  }  void enqueue(QueueType\* q, int item) {  if ((q->rear + 1) % MAX\_VERTICES == q->front) return;  q->rear = (q->rear + 1) % MAX\_VERTICES;  q->queue[q->rear] = item;  }  int dequeue(QueueType\* q) {  if (q->front == q->rear) return;  q->front = (q->front + 1) % MAX\_VERTICES;  return q->queue[q->front];  }  void bfs\_list(GraphType\* g, int v) {  GraphNode\* w;  QueueType q;  q.front = q.rear = 0;  visited[v] = TRUE;  printf("%d 방문 -> ", v);  enqueue(&q, v);  while (!(q.front == q.rear)) {  v = dequeue(&q);  for (w = g->adj\_list[v]; w; w = w->link) {  if (!visited[w->vertex]) {  visited[w->vertex] = TRUE;  printf("%d 방문 -> ", w->vertex);  enqueue(&q, w->vertex);  }  }  }    }  int main() {  GraphType\* g;  g = (GraphType\*)malloc(sizeof(GraphType));  init(g);  int node[12] = { 0, 0, 0, 1, 1, 1, 2, 2, 3, 3, 4, 4 };  int edge[12] = { 1, 2, 3, 0, 2, 4, 0, 1, 0, 4, 1, 3 };  for (int i = 0;i < 5;i++)  insert\_vertex(g, i);  for(int i = 0;i<12;i++)  insert\_edge(g, node[i], edge[i]);  print\_adj\_list(g);  bfs\_list(g, 2);  free(g);  return 0;  }  실행결과    ->  해당 인접 리스트 그래프를 생성 후 2번부터 방문하게 되면  step1. 2번 노드 방문 처리 후 q에 삽입 q = 2  step2. 2번 노드는 1->0 이므로 1번 노드, 2번 노드 순으로 방문 처리 후 q에 삽입 // q = 1, 0  step3. 1번 노드는 4->2->0 이므로 4번 노드를 방문 처리 후 q에 삽입 나머지는 이미 방문 //q=0,4  step4. 0번 노드는 3->2->1 이므로 3번 노드를 방문 처리 후 q에 삽입 나머지는 방문 // q = 4, 3  step5. 4번 노드는 3->1 인데 모두 방문 하였으므로 무시 // q = 3  step6. 3번 노드는 4->0 인데 모두 방문 하였으므로 무시 // q = 0  step7. q가 비었으므로 종료  --> q가 dequeue된 순서로 출력된다.  2 -> 1 -> 0 -> 4 -> 3 |