|  |
| --- |
| Министерство образования Российской Федерации  Пензенский государственный университет  Кафедра «Вычислительная техника» |
| Отчет  по лабораторной работе №6  по курсу «Логика и основы алгоритмизации в инженерных задачах» на тему «Поиск расстояний в графе» |
|  |
|  |
| Выполнила студентка группы 19ВВ3:  Ланцов А.С.  Принял:  Митрохин М. А. |
| Пенза  2020 |

## Поиск в ширину

### **Для матричной формы**

Листинг:

#define \_VALUE\_OF(type,pointer) \*((type\*)pointer)  
#define new(type) malloc(sizeof(type))  
#define new\_const(type,value) &((type){value})  
#define new\_block(type,how\_much) malloc(sizeof(type)\*how\_much)  
#define del(ptr) free(ptr)  
  
#define \_FOR\_(index,range) for(int index=0;index<range;index++)

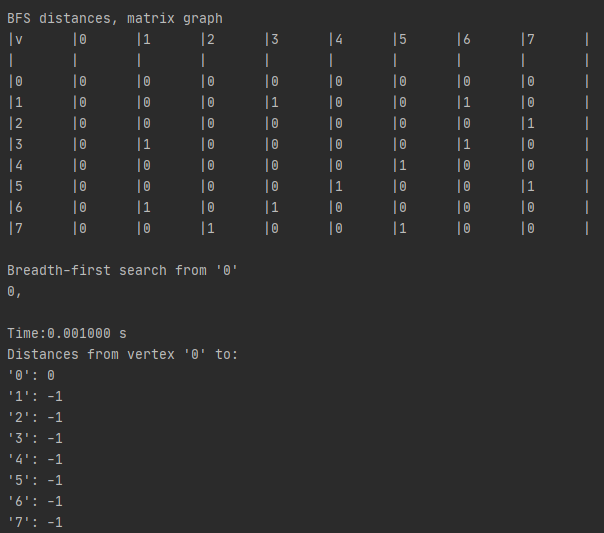
typedef struct numbers\_s {  
 int\* data;  
 size\_t size;  
}numbers\_t;

numbers\_t\* numbers(int size){  
 numbers\_t\* arr = new(numbers\_t);  
 arr->data = new\_block(int,size);  
 arr->size = size;  
  
 return arr;  
}

void fill\_num(numbers\_t\* arr,int value){  
 \_FOR\_(i,arr->size){  
 arr->data[i]=value;  
 }  
}

void test\_bfs\_matrix(int SIZE,int vrtx){  
 printf("\nBFS distances, matrix graph\n");  
 mtrx\_grph\_t\* matrix\_graph = matrixGraph\_create(SIZE);  
  
 srand( time(NULL) );  
 init\_random\_symetric\_matrix(matrix\_graph->matrix,SIZE,0.2);  
 row(matrix\_graph->vertexes,SIZE,0);  
  
 print\_graph(matrix\_graph);  
  
 numbers\_t\* distances = numbers(SIZE);  
 printf("Breadth-first search from '%d'\n",vrtx);  
 clock\_t start\_time, finish\_time;  
  
 bfs\_matrix(matrix\_graph,vrtx,distances);  
  
 double wait\_time = (double)(finish\_time - start\_time) / CLOCKS\_PER\_SEC;  
 printf("Time:%f s\n",wait\_time);  
  
 dist\_log(matrix\_graph->vertexes,vrtx,distances);  
  
 free(distances->data);  
 free(distances);  
}

void bfs\_matrix(mtrx\_grph\_t\* graph,int vertex\_index,numbers\_t\* dist){  
 //every i in dist = -1  
 fill\_num(dist,-1);  
  
 lim\_queue\* vertexes\_queue = lim\_queue\_create(graph->size);  
 lim\_queue\_push(vertexes\_queue,vertex\_index);  
   
 int v;//next vertex  
 int\* front;  
 dist->data[vertex\_index] = 0;  
  
 printf("%d,",vertex\_index);  
 while (vertexes\_queue->length > 0){  
 v = lim\_queue\_pop(vertexes\_queue);  
   
 \_FOR\_(i,graph->size){  
 if (graph->matrix[v][i] && dist->data[i]==-1) {  
 lim\_queue\_push(vertexes\_queue,i);  
   
 dist->data[i]=dist->data[v]+1;  
 printf("%d,",i);  
 }  
 }  
 }  
 printf("\n\n");  
 lim\_queue\_free(vertexes\_queue)  
}

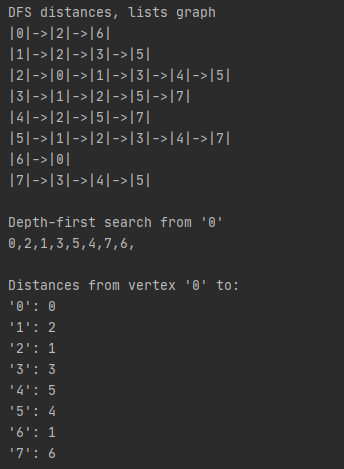
Результат:  


### **Для листовой формы**

Листинг:

void bfs\_lists(list\_graph\_t\* graph,int vertex\_index,numbers\_t\* dist){  
 fill\_num(dist,-1);  
 list\_t\* vertexes\_queue = list\_create();  
 que\_insert(vertexes\_queue,new(int));  
 \_VALUE\_OF(int,vertexes\_queue->tail->value) = vertex\_index;  
  
 int v,u;//next vertex  
 node\_t\* next\_vrtx;  
 dist->data[vertex\_index] = 0;  
 int\* front;  
  
 print\_vertex("%v,",graph->vertexes[vertex\_index]);  
 while (!que\_isempty(vertexes\_queue)){  
 front = que\_remove(vertexes\_queue);  
 v = \_VALUE\_OF(int,front);  
 del(front);  
  
 next\_vrtx = graph->lists[v]->head;  
  
 while(next\_vrtx->next){  
 next\_vrtx = next\_vrtx->next;  
 u = get\_vertex\_index(graph->vertexes,get\_vrtx\_name(next\_vrtx->value),graph->size);  
 if(dist->data[u]==-1){  
 que\_insert(vertexes\_queue,new(int));  
 \_VALUE\_OF(int,vertexes\_queue->tail->value) = u;  
 dist->data[u]=dist->data[v]+1;  
  
 print\_vertex("%v,",graph->vertexes[u]);  
 }  
  
 }  
  
 }  
 printf("\n\n");  
 free(vertexes\_queue);  
}

void test\_bfs\_lists(int SIZE,int vrtx\_index){  
 printf("\nBFS distances, lists graph\n");  
 mtrx\_grph\_t\* matrix\_graph = matrixGraph\_create(SIZE);  
  
 srand( time(NULL) );  
 init\_random\_symetric\_matrix(matrix\_graph->matrix,SIZE,0.2);  
 row(matrix\_graph->vertexes,SIZE,0);  
  
 list\_graph\_t\* list\_graph = matrix\_to\_list(matrix\_graph);  
 print\_list\_graph(list\_graph);  
  
 numbers\_t\* distances = numbers(SIZE);  
 printf("Breadth-first search from ");  
 print\_vertex("'%v'\n",list\_graph->vertexes[vrtx\_index]);  
 bfs\_lists(list\_graph,vrtx\_index,distances);  
 dist\_log(matrix\_graph->vertexes,vrtx\_index,distances);  
}

Результат:  


## Поиск в длину

### **Для матричной формы**

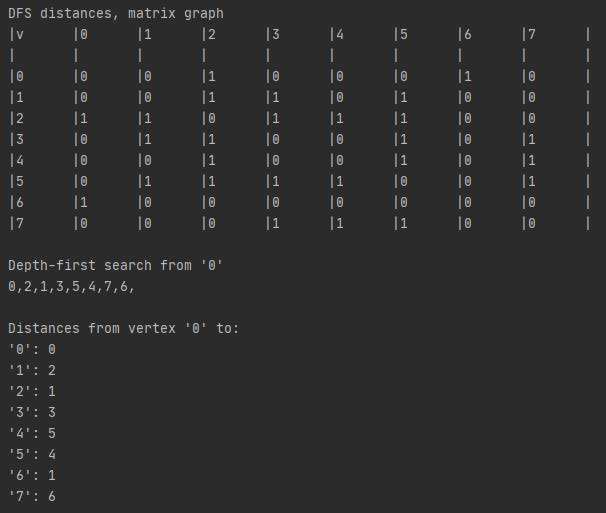
Листинг:

void hardStack\_matrix\_dist(mtrx\_grph\_t\* graph, int v\_index, numbers\_t\* dist){  
 printf("%d,",graph->vertexes[v\_index]);  
 for (int i=0; i<graph->size; i++){  
 if (graph->matrix[v\_index][i] && dist->data[i]==-1){  
 dist->data[i] = dist->data[v\_index]+1;  
 hardStack\_matrix\_dist(graph,i,dist);  
 }  
 }  
}

void dfs\_matrix(mtrx\_grph\_t\* graph, int v\_index, numbers\_t\* dist){  
 fill\_num(dist,-1);  
 dist->data[v\_index] = 0;  
 hardStack\_matrix\_dist(graph,v\_index,dist);  
 printf("\n\n");  
}

void test\_dfs\_matrix(int SIZE, int vrtx){  
 printf("DFS distances, matrix graph\n");  
 mtrx\_grph\_t\* matrix\_graph = matrixGraph\_create(SIZE);  
 srand( time(NULL) );  
  
 init\_random\_symetric\_matrix(matrix\_graph->matrix,SIZE,0.5);  
 row(matrix\_graph->vertexes,SIZE,0);  
  
 print\_graph(matrix\_graph);  
  
 numbers\_t\* distances = numbers(SIZE);  
  
 printf("Depth-first search from '%d'\n",vrtx);  
 dfs\_matrix(matrix\_graph,vrtx,distances);  
 dist\_log(matrix\_graph->vertexes,vrtx,distances);  
}

Результат:



### **Для листовой формы**

void hardStack\_lists\_dist(list\_graph\_t\* graph,int v\_index, numbers\_t\* dist){  
 node\_t\* next = graph->lists[v\_index]->head;  
 int next\_index;  
  
 printf("%d,",graph->vertexes[v\_index].name);  
 for (int i=0; i<graph->lists[v\_index]->depth; i++){  
 next\_index=get\_vertex\_index(graph->vertexes,get\_vrtx\_name(next->value),graph->size);  
 if (dist->data[next\_index]==-1) {  
 dist->data[next\_index] = dist->data[v\_index]+1;  
 hardStack\_lists\_dist(graph, next\_index,dist);  
 }  
 next = next->next;  
 }  
}

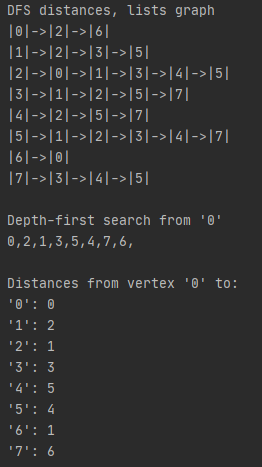
void dfs\_lists(list\_graph\_t\* graph, int v\_index, numbers\_t\* dist){  
 fill\_num(dist,-1);  
 dist->data[v\_index] = 0;  
 hardStack\_lists\_dist(graph,v\_index,dist);  
 printf("\n\n");  
}

void test\_dfs\_lists(int SIZE, int vrtx){

printf("\nDFS distances, lists graph\n");  
 mtrx\_grph\_t\* matrix\_graph = matrixGraph\_create(SIZE);  
  
 srand( time(NULL) );  
 init\_random\_symetric\_matrix(matrix\_graph->matrix,SIZE,0.5);  
 row(matrix\_graph->vertexes,SIZE,0);  
  
 list\_graph\_t\* list\_graph = matrix\_to\_list(matrix\_graph);  
 print\_list\_graph(list\_graph);  
  
 numbers\_t\* distances = numbers(SIZE);  
 printf("Depth-first search from '%d'\n",vrtx);  
 dfs\_lists(list\_graph,vrtx,distances);  
 dist\_log(matrix\_graph->vertexes,vrtx,distances);

}

Результат:



## Приложение

Структуры:  
typedef struct node{  
 uint8\_t\* value;  
  
 struct node\* last;  
 struct node\* next;  
}node\_t;  
  
typedef struct list{  
 node\_t\* head;  
 node\_t\* tail;  
  
 int depth;  
  
}list\_t;

typedef struct matrix\_graph{  
 int\*\* src;  
  
 int\*\* matrix;  
 char\*\* vertexes;  
 int size;  
  
}mtrx\_grph\_t;

typedef struct list\_graph{  
 list\_t\*\* lists;  
 uint8\_t\*\* vertexes;  
  
 int size;  
}list\_graph\_t;

typedef struct vertex\_s{  
 int name;  
 char color;  
}vertex;  
  
#define get\_vrtx\_color(pointer) (\*((vertex\*)pointer)).color  
#define get\_vrtx\_name(pointer) (\*((vertex\*)pointer)).name

Очередь:

int que\_isempty(list\_t\* queue){  
 if(queue->head==0)  
 return 1;  
 else  
 return 0;  
}

void que\_insert(list\_t\* queue, void\* value) {  
 queue->tail = push(queue->tail,value);  
 if(queue->head==0) {  
 queue->head = queue->tail;  
 }  
  
 queue->depth ++;  
}  
  
void\* que\_remove(list\_t\* queue) {  
 if(que\_isempty(queue)) {  
 return 0;  
 }  
 else{  
 node\_t\* temp;  
 void\* head\_value;  
  
 head\_value = queue->head->value;  
 temp = queue->head;  
 queue->head = queue->head->next;  
  
  
 if(!queue->head){//if queue is empty now  
 queue->tail=NULL;  
 }  
 else{  
 queue->head->last = NULL;  
 }  
  
 free(temp);  
  
 queue->depth --;  
 return head\_value;  
 }  
  
}

typedef struct limited\_queue\_s {  
 int\* data;  
 size\_t first;  
 size\_t last;  
 size\_t value\_size;  
 size\_t length;  
 size\_t limit;  
  
}lim\_queue;  
  
#define lim\_queue\_free(q) \  
free(q->data);\  
free(q);

lim\_queue\* lim\_queue\_create(size\_t limit){  
 lim\_queue\* queue = malloc(sizeof(lim\_queue));  
  
 queue->data = malloc(sizeof(int)\*limit);  
 queue->limit = limit;  
 queue->length = 0;  
 queue->first = 0;  
 queue->last = -1;  
  
 return queue;  
}  
  
int lim\_queue\_push(lim\_queue\* queue,int value){  
 queue->length ++;  
  
 if (queue->length <= queue->limit){  
 queue->last++;  
 queue->data[queue->last]=value;  
 return 1;  
 }  
 else{  
 return 0;  
 }  
}  
  
int lim\_queue\_pop(lim\_queue\* queue){  
 queue->length --;  
  
 if (queue->length >= 0){  
 queue->first++;  
 return queue->data[queue->first-1];  
 }  
 else{  
 return NULL;  
 }  
}