|  |
| --- |
| Министерство образования Российской Федерации  Пензенский государственный университет  Кафедра «Вычислительная техника» |
| Отчет  по лабораторной работе №7  по курсу «Логика и основы алгоритмизации в инженерных задачах»  на тему «Поиск расстояний во взвешенном графе» |
|  |
|  |
| Выполнила студентка группы 19ВВ3:  Ланцов А.С.  Принял:  Митрохин М. А. |
| Пенза  2020 |

# **Задание 1**

## Неориентированный взвешенный граф

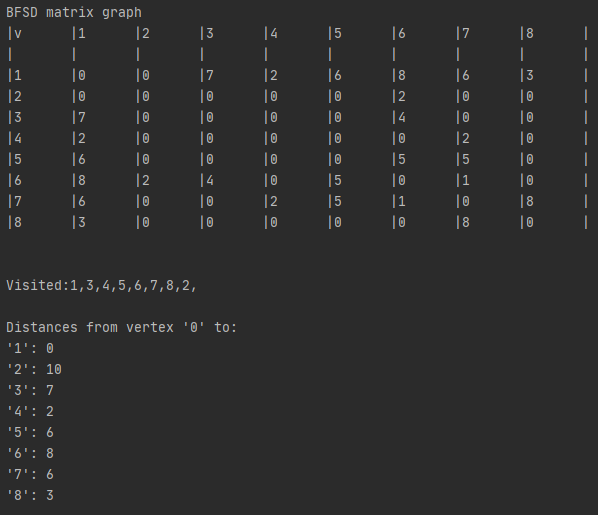
Листинг:

void init\_matrix(mtrx\_grph\_t\* graph, int MAX, float chance, char\*\* args){  
 int wt = strcmp(args[2], "wt");  
 if (strcmp(args[1], "ornd") == 0) {  
 if(wt == 0){init\_random\_oriented\_matrix(graph->matrix, graph->size, MAX, chance);}  
 else {init\_bin\_oriented\_matrix(graph->matrix, graph->size, chance);}  
 }  
 else{  
 if(wt == 0){init\_random\_symetric\_matrix(graph->matrix, graph->size, MAX, chance);}  
 else{init\_bin\_symetric\_matrix(graph->matrix, graph->size, chance);}  
 }  
}

void test\_bfsd\_matrix(int SIZE,int MAX,int argc, char\*\* argv){  
 if (argc > 2) {  
 printf("\nBFSD matrix graph\n");  
 mtrx\_grph\_t \*matrix\_graph = matrixGraph\_create(SIZE);  
 srand(time(NULL));  
 row\_vertexes(matrix\_graph->vertexes, SIZE, 1);  
  
 init\_matrix(matrix\_graph,MAX,0.5,argv);  
  
 print\_graph(matrix\_graph);  
 //print\_matrix(stdout, matrix\_graph->matrix, SIZE, SIZE, ',');  
  
 numbers\_t \*distances = numbers(SIZE);  
 bfsd\_matrix(matrix\_graph, 0, distances);  
 dist\_log(matrix\_graph->vertexes, 0, distances);  
 }  
 else{printf("Two arguments expected");}  
}

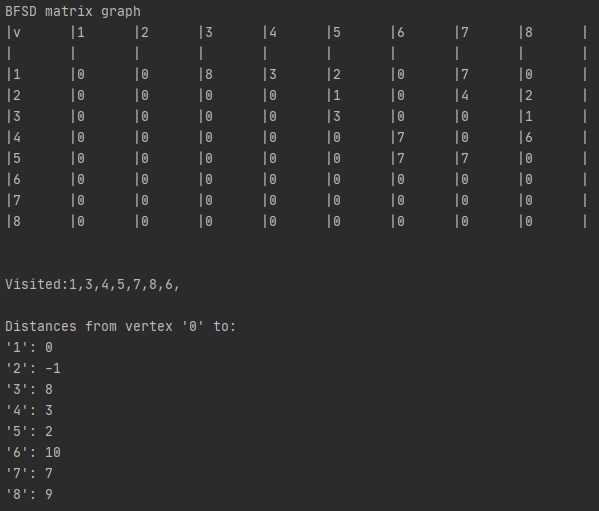
void bfsd\_matrix(mtrx\_grph\_t\* graph,int vertex\_index,numbers\_t\* dist){  
 fill\_num(dist,-1);  
  
 lim\_queue\* vertexes\_queue = lim\_queue\_create(graph->size);  
 lim\_queue\_push(vertexes\_queue,vertex\_index);  
  
 int v;//next vertex  
 int\* front;  
 dist->data[vertex\_index] = 0;  
  
 print\_vertex("\nVisited:%v,",graph->vertexes[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]+graph->matrix[v][i];  
 print\_vertex("%v,",graph->vertexes[i]);  
 }  
 }  
 }  
 printf("\n\n");  
 lim\_queue\_free(vertexes\_queue);  
}

Результат:



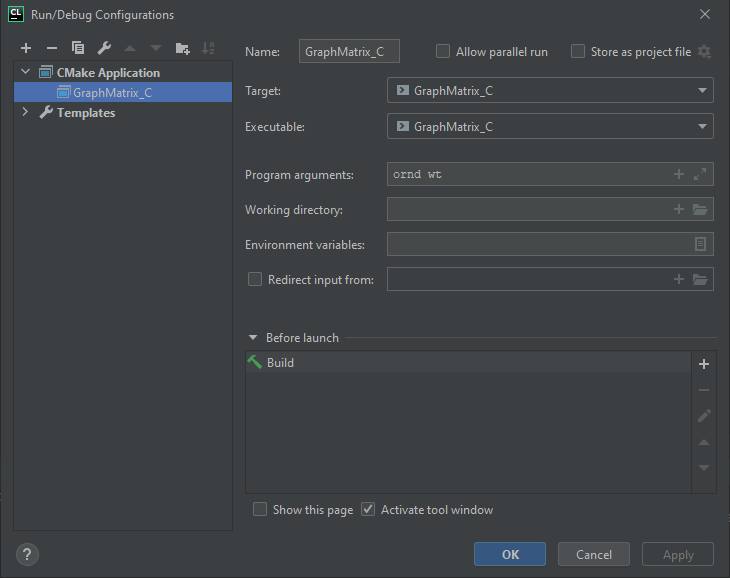
# **Задание 2**

## Ориентированный взвешенный граф



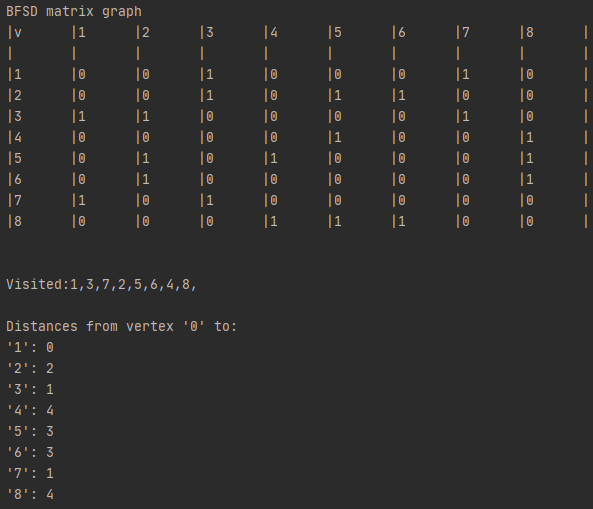
# **Задание 3**

Параметры программы были заданы с помощью среды разработки

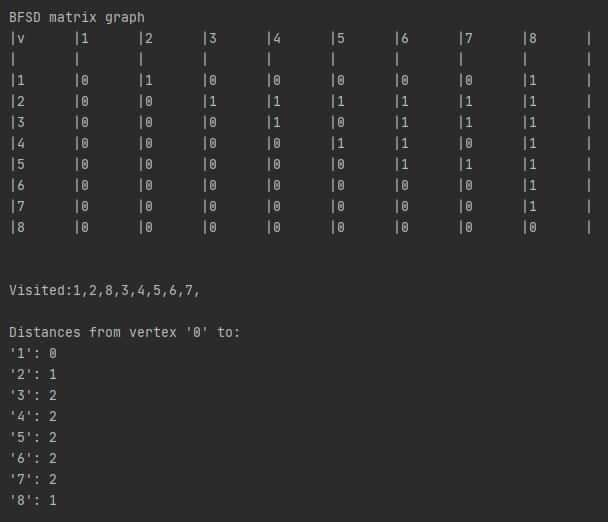


|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| ornd | ориентированный |
| nornd | неориентированный |
| wt | взвешенный |
| nwt | невзвешенный |

## Неориентированный невзвешенный граф (поиск в ширину)



## Ориентированный невзвешенный граф



# **Приложение**

Макросы

#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++)

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;  
 }  
}