Đề tài: tìm đường đi ngắn nhất trên bản đồ HUST

GVHD: Trần Thị Thanh Hải

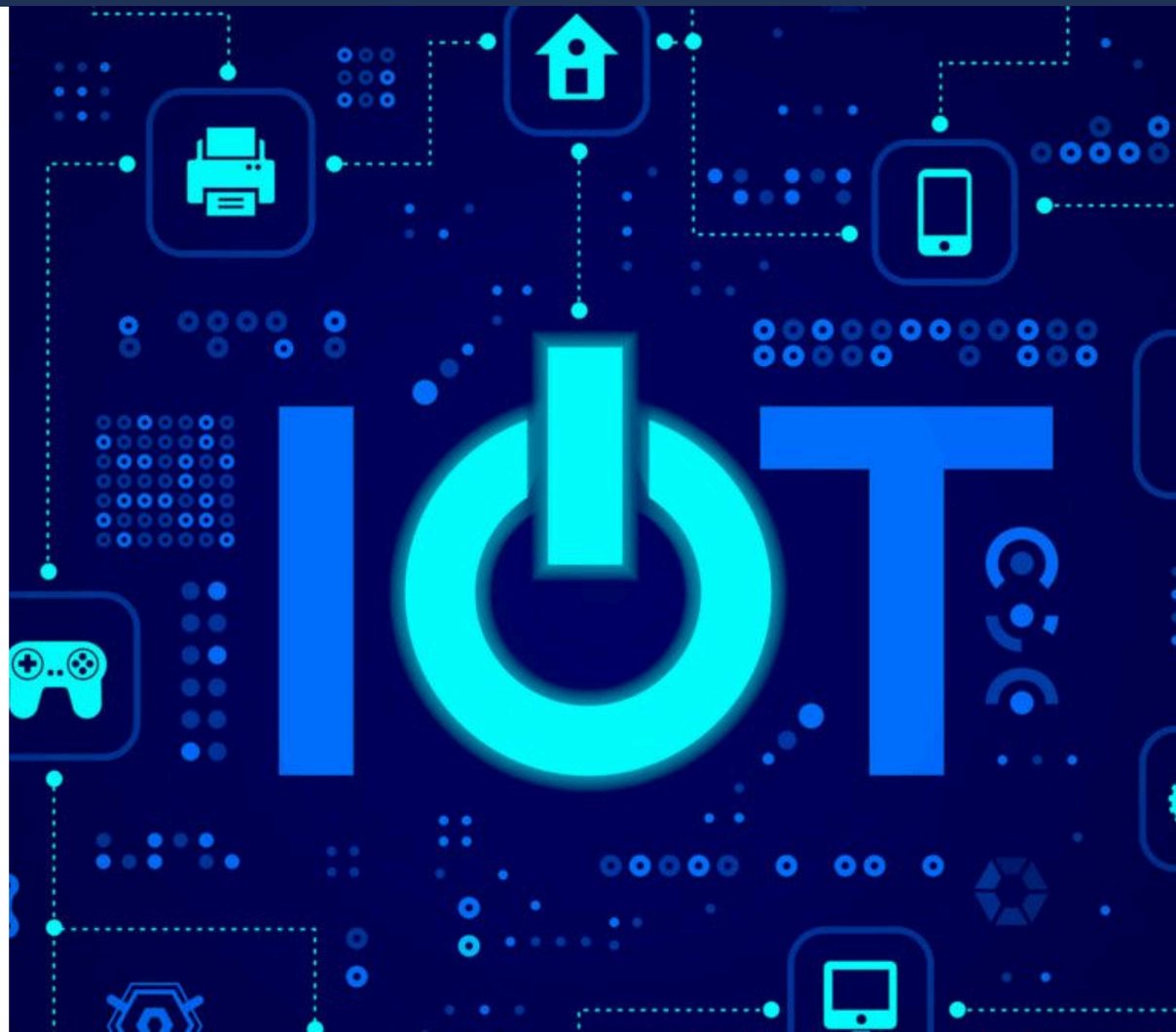
TỔNG QUAN VỀ PROJECT

01 Giới thiệu thành viên

02 Đề tài

03 Phân chia công việc

04 Code chương trình



Thành viên nhóm

01

Lê Anh Đức
20233324

02

Nguyễn Tiến Đạt
20233307

03

Đỗ Việt Thành Đô
20233317

Nhiệm vụ của mỗi thành viên

Đức

Lên ý tưởng
Code chính (60%)

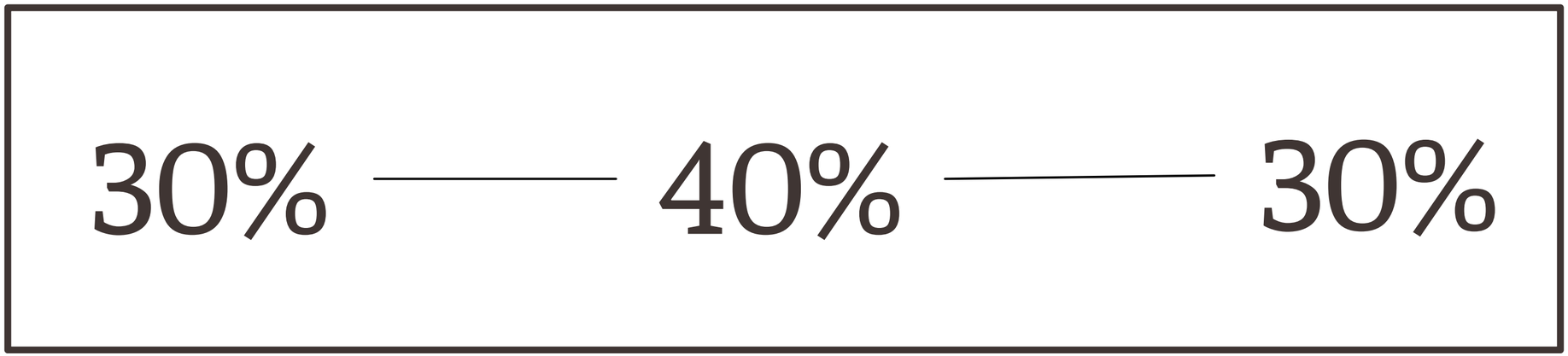
Đạt

Code (20%)
Làm slide báo cáo

Đô

Lên ý tưởng
Code (20%)

Trọng số đóng góp cho toàn bộ project



“Biểu diễn bản đồ HUST bằng đồ thị (đỉnh: các địa điểm; cạnh: các lối đi có trọng số). Dùng danh sách kề để lưu dữ liệu. Áp dụng min heap để tìm đường đi ngắn nhất bằng Dijkstra.”

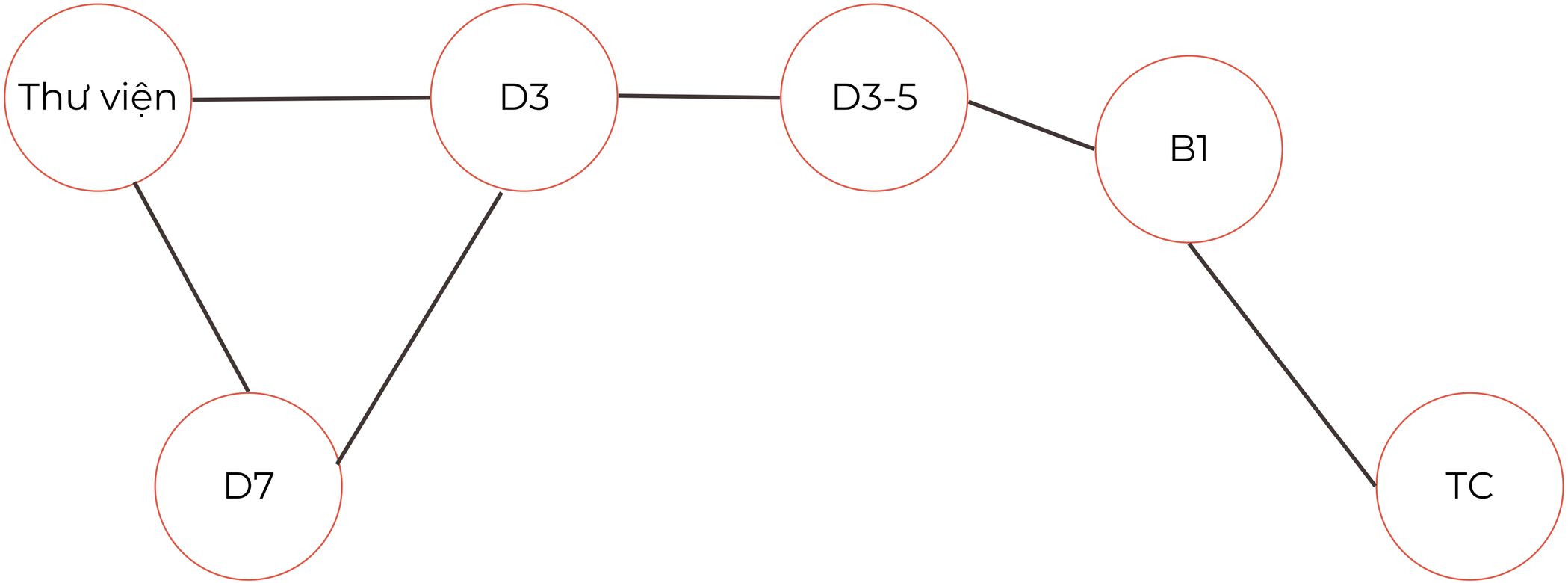
—Ý tưởng

Các phần kiến thức được sử dụng

- Danh sách liên kết
- Đồ thị
- Đệ quy
- Mảng
- Thuật toán tìm đường đi ngắn nhất diijkstra
- hàng đợi ưu tiên theo min heap

Bản đồ sử dụng

Một số tòa được đưa vào project



Code chương trình

1. Khai báo thư viện

```c
#include <stdio.h>      // Thư viện cho hàm nhập xuất (printf, scanf,...)
#include <stdlib.h>     // Cho các hàm cấp phát bộ nhớ (malloc, free,...)
#include <string.h>     // Xử lý chuỗi (strcmp, strcpy,...)
#include <ctype.h>      // Xử lý ký tự (toupper, isdigit,...)
#include <limits.h>     // Định nghĩa giá trị INT_MAX (giá trị lớn nhất của int)
```

Quay lại chương trình làm việc

Code chương trình

2. Cấu trúc node danh sách kề

```
7    struct node {
8        int dinh;
9        int trong_so;
10       struct node *next;
11   };
12   typedef struct node node;
```

Giải thích:

- **Biểu diễn một đỉnh kề trong đồ thị.**
- **dinh: số hiệu đỉnh**
- **trong_so: độ dài đoạn đường (trọng số cạnh).**
- **next: trỏ tới node kề tiếp theo trong danh sách.**

Quay lại chương trình làm việc

Code chương trình

3. Hàm tạo node

```
14  node *createnode(int dinh, int trong_so) {
15      node *newnode = (node*)malloc(sizeof(node)); // Cấp phát bộ nhớ
16      newnode->dinh = dinh;                         // Gán số hiệu đỉnh
17      newnode->trong_so = trong_so;                 // Gán trọng số
18      newnode->next = NULL;                         // Node cuối => next = NULL
19      return newnode;
20  }
```

- **Tạo node để thêm vào danh sách kề.**

Quay lại chương trình làm việc

Code chương trình

4. Cấu trúc đồ thị

```
22    struct Dothi {
23        int so_dinh;
24        node **danh_sach_ke;
25    };
26    typedef struct Dothi Dothi;
```

- so_dinh: số lượng đỉnh trong đồ thị.
- danh_sach_ke: mảng con trỏ tới danh sách liên kết (kề) của từng đỉnh.

- Ví dụ: danh_sach_ke[0] là danh sách các đỉnh kề với đỉnh 0.

Quay lại chương trình làm việc

Code chương trình

5. Tạo đồ thị

```
33   // Khoi tao Do thi
34   Dothi *createDothi(int so_dinh){
35       // Cap phat dong cho do thi
36       Dothi *newgraph = (Dothi*)malloc(sizeof(Dothi));
37       // so dinh cho do thi moi
38       newgraph->so_dinh = so_dinh;
39       // Cap phat dong de tao ma tran ke bang mang 2 chieu
40       newgraph->danh_sach_ke = (node**)malloc(so_dinh * sizeof(node*));
41       // Khoi tao cac phan tu trong mang danh sach ke = NULL
42       for(int i=0;i<so_dinh;i++){
43           newgraph->danh_sach_ke[i] = NULL;
44       }
45       return newgraph;
46   }
```

- **Tạo đồ thị mới có so_dinh đỉnh, khởi tạo danh sách kề mỗi đỉnh bằng NULL.**

Quay lại chương trình làm việc

Code chương trình

6. Thêm cạnh vào đồ thị

```
48  // Them canh vao do thi vo huong
49  void them_canh(Dothi *dothi, int dinh_nguon, int dinh_dich, int trong_so){
50      // Them dinh dich vao danh sach ke cua dinh nguon
51      node *newgraph = createnode(dinh_dich,trong_so);
52      newgraph->next = dothi->danh_sach_ke[dinh_nguon];
53      dothi->danh_sach_ke[dinh_nguon] = newgraph;
54      // Them dinh nguon vao danh sach ke cua dinh dich
55      newgraph = createnode(dinh_nguon,trong_so);
56      newgraph->next = dothi->danh_sach_ke[dinh_dich];
57      dothi->danh_sach_ke[dinh_dich] = newgraph;
58  }
```

- **Thêm cạnh 2 chiều (vì đồ thị là vô hướng) giữa dinh_nguon và dinh_dich**

Quay lại chương trình làm việc

Code chương trình

7. Cấu trúc node min heap (Hàng đợi ưu tiên)

```
struct node_min_heap{
    int dinh;
    int khoang_cach;
};
typedef struct node_min_heap node_min_heap;
```

- Lưu trữ đỉnh và khoảng cách tạm thời trong quá trình chạy Dijkstra.

Quay lại chương trình làm việc

Code chương trình

7. Cấu trúc node min heap (Hàng đợi ưu tiên)

Min Heap là cấu trúc dữ liệu quan trọng dùng để:

- **Luôn nhanh chóng lấy đỉnh có khoảng cách nhỏ nhất chưa được xử lý trong thuật toán Dijkstra.**
- **Hỗ trợ thao tác extractMin() và decreaseKey() nhanh chóng (O(log V)).**

```
39  struct node_min_heap {
40      int dinh;           // Đỉnh trong đồ thị
41      int khoang_cach;    // Khoảng cách hiện tại từ đỉnh nguồn
42  };
43
```

Mỗi node lưu:

- **dinh: số hiệu đỉnh trong đồ thị.**
- **khoang_cach: giá trị khoảng cách tạm thời trong thuật toán Dijkstra.**

Quay lại chương trình làm việc

Code chương trình

7. Cấu trúc node min heap (Hàng đợi ưu tiên)

```
69    struct min_heap{
70        int kich_thuoc;          // so luong phan tu hien tai
71        int suc_chua;            // suc chua toi da
72        int *vitri;              // vi tri dinh trong min heap
73        node_min_heap **arr;     // mang tro den cac node trong heap
74    };
75    typedef struct min_heap min_heap;
```

- **kich_thuoc: dùng để biết còn bao nhiêu đỉnh chưa được xử lý.**
- **vitri[v]: giữ vị trí của đỉnh v trong mảng arr để hỗ trợ giảm key nhanh.**
- **arr[]: chính là heap — phần tử nhỏ nhất luôn nằm ở arr[0].**

Quay lại chương trình làm việc

Code chương trình

8. Tạo node minheap

```
77    // tao node min heap
78    node_min_heap *taoNodeMinHeap(int dinh, int khoang_cach) {
79        node_min_heap *node = (node_min_heap*)malloc(sizeof(node_min_heap));
80        node->dinh = dinh;
81        node->khoang_cach = khoang_cach;
82        return node;
83    }
```

- **Cấp phát động và khởi tạo 1 node trong heap**

Quay lại chương trình làm việc

Code chương trình

9. Tạo Min Heap

```
86  min_heap *taoMinHeap(int suc_chua) {
87      min_heap *minheap = (min_heap*)malloc(sizeof(min_heap));
88      minheap->kich_thuoc = 0;
89      minheap->suc_chua = suc_chua;
90      minheap->vitri = (int*)malloc(suc_chua * sizeof(int));
91      minheap->arr = (node_min_heap**)malloc(suc_chua * sizeof(node_min_heap*));
92      return minheap;
93  }
```

Khởi tạo Min Heap với:

- **Dung lượng chứa đủ suc_chua phần tử.**
- **Mảng vitri[] ban đầu sẽ được gán sau.**

Quay lại chương trình làm việc

Code chương trình

10. Hàm đổi chỗ swap 2 node heap

```
95    // Hoan doi
96    void swapNode(node_min_heap **a, node_min_heap **b) {
97        node_min_heap *temp = *a;
98        *a = *b;
99        *b = temp;
100   }
```

Dùng trong sắp xếp heap, để hoán đổi hai node.

Quay lại chương trình làm việc

Code chương trình

11. Hàm ShiftDown để duy trì tính chất Min Heap

a. Xác định vị trí ban đầu cần xử lý

```
103   // Moi node cha phai nho hon hoac bang cac node con cua no
104   // node nho nhat luon o goc root
105   void shiftDown(min_heap *minheap, int idx) {
106       int nho_nhat = idx;
107       // tinh chi so con trai
108       int trai = 2 * idx + 1;
109       // Tinh chi so con phai
110       int phai = 2 * idx + 2;
```

- nho_nhat giữ chỉ số của node nhỏ nhất hiện tại (bắt đầu là node cha ở idx).
- trai, phai lần lượt là chỉ số của con trái và con phải trong mảng heap (cây nhị phân lưu trong mảng).

Quay lại chương trình làm việc

Code chương trình

11. Hàm ShiftDown để duy trì tính chất Min Heap

b. So sánh node cha với con trái

```
112     if (trai < minheap->kich_thuoc &&
113         minheap->arr[trai]->khoang_cach < minheap->arr[nho_nhat]->khoang_cach)
114         nho_nhat = trai;
```

- Kiểm tra xem con trái có tồn tại (trai < kich_thuoc).
- So sánh khoảng cách (giá trị để heap sắp xếp) của con trái với node cha.
- Nếu con trái nhỏ hơn, cập nhật nho_nhat thành con trái.

Quay lại chương trình làm việc

Code chương trình

11. Hàm ShiftDown để duy trì tính chất Min Heap

c. So sánh node cha với con phải

```
116     if (phai < minheap->kich_thuoc &&
117         minheap->arr[phai]->khoang_cach < minheap->arr[nho_nhat]->khoang_cach)
118         nho_nhat = phai;
```

- Kiểm tra con phải tồn tại.
- So sánh khoảng cách của con phải với node hiện tại nhỏ nhất.
- Cập nhật nho_nhat nếu con phải nhỏ hơn.

Quay lại chương trình làm việc

Code chương trình

11. Hàm ShiftDown để duy trì tính chất Min Heap

d. Hoán đổi nếu cần và tiếp tục đệ quy

```
if (nho_nhat != idx) {
    // Cap nhat vi tri
    minheap->vitri[minheap->arr[nho_nhat]->dinh] = idx;
    minheap->vitri[minheap->arr[idx]->dinh] = nho_nhat;

    // Thuc hien hoan doi
    swapNode(&minheap->arr[nho_nhat], &minheap->arr[idx]);
    // Goi lai de dua cay ve Min Heap ( cha nho hon con)
    shiftDown(minheap, nho_nhat);
}
```

- Nếu node nhỏ nhất không phải node cha ban đầu, ta phải hoán đổi node cha với node nhỏ nhất trong các node con.
- Cập nhật mảng vitri để ghi nhận đúng vị trí mới của các node trong heap.
- Gọi đệ quy shiftDown tại vị trí mới nho_nhat để đảm bảo cây con phía dưới vẫn giữ tính chất Min Heap.

Quay lại chương trình làm việc

Code chương trình

12. Kiểm tra xem Min Heap có đang rỗng hay không

```
132   // Kiem tra Heap co rong khong
133   int isEmpty(min_heap *minheap) {
134       return minheap->kich_thuoc == 0;
135   }
```

- **minheap là con trỏ trỏ tới cấu trúc Min Heap.**
- **minheap->kich_thuoc là số lượng phần tử hiện tại trong heap.**

Quay lại chương trình làm việc

Code chương trình

13. lấy ra node có khoảng cách nhỏ nhất trong Min Heap và loại bỏ node đó khỏi heap.

```
137  // Lay node co khoang cach nho nhat va loại bo no khoi Heap
138  node_min_heap *extractMin(min_heap *minheap) {
139      // Neu danh sach rong thi dung ham
140      if (isEmpty(minheap))
141          return NULL;
142
143      node_min_heap *root = minheap->arr[0];
144      node_min_heap *cuoi = minheap->arr[minheap->kich_thuoc - 1];
145      minheap->arr[0] = cuoi;
146
147      // Cap nhat vi tri
148      minheap->vitri[root->dinh] = minheap->kich_thuoc - 1;
149      minheap->vitri[cuoi->dinh] = 0;
150
151      minheap->kich_thuoc--;
152      shiftDown(minheap, 0);
153      return root;
154  }
```

- Kiểm tra heap rỗng, trả về NULL nếu rỗng.
- Lấy node nhỏ nhất ở vị trí 0.
- Thay node gốc bằng node cuối cùng trong heap.
- Cập nhật vị trí 2 node trong mảng vitri.
- Giảm kích thước heap đi 1.
- Gọi shiftDown để duy trì cấu trúc Min Heap.
- Trả về node nhỏ nhất vừa lấy ra.

Quay lại chương trình làm việc

Code chương trình

13. lấy ra node có khoảng cách nhỏ nhất trong Min Heap và loại bỏ node đó khỏi heap.

```
137  // Lay node co khoang cach nho nhat va loại bo no khoi Heap
138  node_min_heap *extractMin(min_heap *minheap) {
139      // Neu danh sach rong thi dung ham
140      if (isEmpty(minheap))
141          return NULL;
142
143      node_min_heap *root = minheap->arr[0];
144      node_min_heap *cuoi = minheap->arr[minheap->kich_thuoc - 1];
145      minheap->arr[0] = cuoi;
146
147      // Cap nhat vi tri
148      minheap->vitri[root->dinh] = minheap->kich_thuoc - 1;
149      minheap->vitri[cuoi->dinh] = 0;
150
151      minheap->kich_thuoc--;
152      shiftDown(minheap, 0);
153      return root;
154  }
```

- Kiểm tra heap rỗng, trả về NULL nếu rỗng.
- Lấy node nhỏ nhất ở vị trí 0.
- Thay node gốc bằng node cuối cùng trong heap.
- Cập nhật vị trí 2 node trong mảng vitri.
- Giảm kích thước heap đi 1.
- Gọi shiftDown để duy trì cấu trúc Min Heap.
- Trả về node nhỏ nhất vừa lấy ra.

Quay lại chương trình làm việc

Code chương trình

14. Giảm giá trị khoảng cách và duy trì tính chất Min Heap bằng cách shift-up

```
156  // Giảm giá trị khoảng cách của một đỉnh và shift-up để duy trì tính chất min heap
157  void decreaseKey(min_heap *minheap, int dinh, int khoang_cach) {
158      int i = minheap->vitri[dinh];
159      minheap->arr[i]->khoang_cach = khoang_cach;
160
161      while (i && minheap->arr[i]->khoang_cach < minheap->arr[(i - 1) / 2]->khoang_cach) {
162          // Cap nhat vi tri
163          minheap->vitri[minheap->arr[i]->dinh] = (i - 1) / 2;
164          minheap->vitri[minheap->arr[(i - 1) / 2]->dinh] = i;
165
166          // Hoan doi
167          swapNode(&minheap->arr[i], &minheap->arr[(i - 1) / 2]);
168          i = (i - 1) / 2;
169      }
170  }
```

- Cập nhật khoảng cách mới cho đỉnh trong Min Heap
- So sánh với đỉnh cha để giữ tính chất Min Heap
- Nếu nhỏ hơn đỉnh cha, hoán đổi v trí với đỉnh cha
- Tiếp tục lặp lại đến khi đỉnh không còn nhỏ hơn cha hoặc lên đến gốc heap

Quay lại chương trình làm việc

Code chương trình

15. Kiểm tra xem một đỉnh (node) có đang nằm trong Min Heap hay không.

```
172    // Kiem tra dinh con trong Min Heap khong
173    int isInMinHeap(min_heap *minheap, int dinh) {
174        return minheap->vitri[dinh] < minheap->kich_thuoc;
175    }
```

- Mỗi đỉnh có chỉ số vị trí lưu trong vitri[dinh].
- So sánh chỉ số đó với kich_thuoc (số phần tử hiện tại của heap).
- Nếu vitri[dinh] < kich_thuoc → đỉnh còn trong heap

Quay lại chương trình làm việc

Code chương trình

16. Hàm giải phóng bộ nhớ của MinHeap

```
177   // Hàm giải phóng bộ nhớ của MinHeap
178   void giaiPhongMinHeap(min_heap* minHeap) {
179       if (minHeap == NULL) return;
180
181       for (int i = 0; i < minHeap->kich_thuoc; i++) {
182           if (minHeap->arr[i] != NULL) {
183               free(minHeap->arr[i]);
184           }
185       }
186       free(minHeap->arr);
187       free(minHeap->vitri);
188       free(minHeap);
189   }
```

- Kiểm tra minHeap có tồn tại không.
- Duyệt qua từng phần tử trong minHeap->arr:

-> Nếu phần tử khác NULL → dùng free() để giải phóng.

- Giải phóng mảng arr chứa các node.
- Giải phóng mảng vitri lưu vị trí đỉnh.
- Cuối cùng giải phóng chính struct minHeap.

Quay lại chương trình làm việc

Code chương trình

17. Thuật toán Dijkstra tìm đường đi ngắn nhất

a. Khởi tạo Min Heap và các mảng

```
193   void dijkstra(Dothi* doThi, int dinhNguon, int khoangCach[], int parent[]) {
194       int soDinh = doThi->so_dinh;
195       min_heap* minHeap = taoMinHeap(soDinh);
```

- Lấy số lượng đỉnh của đồ thị.
- Tạo Min Heap có kích thước tương ứng với số đỉnh.

Quay lại chương trình làm việc

Code chương trình

17. Thuật toán Dijkstra tìm đường đi ngắn nhất

b. Gán giá trị khởi đầu cho khoangCach và parent

```
197     for (int v = 0; v < soDinh; v++) {
198         khoangCach[v] = INT_MAX;
199         parent[v] = -1;
200         minHeap->arr[v] = taoNodeMinHeap(v, khoangCach[v]);
201         minHeap->vitri[v] = v;
202     }
```

- Gán khoảng cách từ đỉnh nguồn đến tất cả các đỉnh khác là vô cùng (INT_MAX).
- parent[v] = -1: chưa có cha, dùng để truy vết đường đi sau này.
- Tạo node Min Heap tương ứng và lưu vào arr[] và vitri[] (bảng vị trí trong heap)

Quay lại chương trình làm việc

Code chương trình

17. Thuật toán Dijkstra tìm đường đi ngắn nhất

c. Gán khoảng cách đỉnh nguồn = 0 và cập nhật Min Heap

```
khoangCach[dinhNguon] = 0;
decreaseKey(minHeap, dinhNguon, khoangCach[dinhNguon]);
minHeap->kich_thuoc = soDinh;
```

- Vì đỉnh nguồn là điểm xuất phát nên khoangCach = 0.
- Gọi decreaseKey để đẩy đỉnh nguồn lên đầu heap (vì nhỏ nhất).
- Đặt kích thước heap bằng số đỉnh.

Quay lại chương trình làm việc

Code chương trình

17. Thuật toán Dijkstra tìm đường đi ngắn nhất

d. Vòng lặp chính của thuật toán Dijkstra

```
while (!isEmpty(minHeap)) {
    node_min_heap* nodeMin = extractMin(minHeap);
    int u = nodeMin->dinh;
    free(nodeMin); // Giải phóng node sau khi lấy ra khỏi heap
```

- Mỗi lần lặp, lấy ra đỉnh có khoảng cách nhỏ nhất từ heap.
- u là đỉnh đang xét.
- Giải phóng node đã lấy ra để tránh rò rỉ bộ nhớ.

Quay lại chương trình làm việc

Code chương trình

17. Thuật toán Dijkstra tìm đường đi ngắn nhất

e. Duyệt tất cả các đỉnh kề u

```
node* ke = doThi->danh_sach_ke[u];
while (ke != NULL) {
    int v = ke->dinh;
```

- Duyệt danh sách kề của u, lấy từng đỉnh v được nối với u

Quay lại chương trình làm việc

Code chương trình

17. Thuật toán Dijkstra tìm đường đi ngắn nhất

f. Kiểm tra và cập nhật đường đi ngắn hơn

```
if (isInMinHeap(minHeap, v) && khoangCach[u] != INT_MAX &&
    ke->trong_so + khoangCach[u] < khoangCach[v]) {
    khoangCach[v] = khoangCach[u] + ke->trong_so;
    parent[v] = u; // Cập nhật đỉnh cha của v là u
    decreaseKey(minHeap, v, khoangCach[v]);
}
ke = ke->next;
```

-> Đảm bảo luôn tìm được đường đi ngắn nhất từ đỉnh nguồn đến mọi đỉnh khác

- Kiểm tra điều kiện:
  + Đỉnh v chưa được xử lý (vẫn trong Min Heap).
  + Có đường đi tới u (khoangCach[u] khác INT_MAX).
  + Đường đi từ nguồn → u → v ngắn hơn đường hiện tại đến v.

- Nếu thoả mãn:
  + Cập nhật khoảng cách mới đến v.
  + Lưu u là cha của v để truy vết đường đi.
  + Gọi decreaseKey để điều chỉnh vị trí v trong Min Heap (vì khoảng cách đã nhỏ hơn).

Quay lại chương trình làm việc

Code chương trình

18. Hàm in đường đi

```
228   // Hàm in đường đi (đệ quy)
229   void inDuongDi(int parent[], int j, char* tenToaNha[]) {
230       if (parent[j] == -1) {
231           printf("%s", tenToaNha[j]);
232           return;
233       }
234       inDuongDi(parent, parent[j], tenToaNha);
235       printf(" -> %s", tenToaNha[j]);
236   }
```

- **parent[]: lưu đỉnh cha trên đường đi.**
- **j : đỉnh cần in.**
- **Nếu parent[j] == -1: in tên đỉnh j (điểm bắt đầu).**
- **Ngược lại, đệ quy in đường đi đến parent[j].**
- **Rồi in " -> " và tên đỉnh j.**
- **Kết quả: in đường đi từ đầu đến đích.**

Quay lại chương trình làm việc

Code chương trình

19. Hàm giải phóng bộ nhớ

```
238  //-------------------- Giai phong vung nho------------//
239  // Hàm giải phóng bộ nhớ của đồ thị
240  void giaiPhongDoThi(Dothi* doThi) {
241      if (doThi == NULL) return;
242
243      for (int i = 0; i < doThi->so_dinh; i++) {
244          node* hienTai = doThi->danh_sach_ke[i];
245          while (hienTai != NULL) {
246              node* temp = hienTai;
247              hienTai = hienTai->next;
248              free(temp);
249          }
250      }
251      free(doThi->danh_sach_ke);
252      free(doThi);
253  }
```

-Kiểm tra đồ thị null: Nếu doThi == NULL thì thoát ra luôn.

-Duyệt từng đỉnh:

- Với mỗi đỉnh, duyệt qua danh sách liên kết các đỉnh kề.
- Giải phóng từng node trong danh sách.

-Giải phóng mảng danh sách kề: Sau khi đã giải phóng các node riêng lẻ.

-Giải phóng struct đồ thị: Cuối cùng, giải phóng bản thân doThi.

Quay lại chương trình làm việc

Code chương trình

20. Thông tin khoảng cách giữa các tòa

```
255  //---------- Thong tin khoang cach giua cac toa----------//
     You, 3 days ago | 1 author (You)
256  typedef struct {
257      const char *toa1;
258      const char *toa2;
259      int khoang_cach; // mét
260  } Canh;
261
262  Canh ds_canh[] = {
263      {"D3", "D3-5", 20},
264      {"D3-5", "B1", 100},
265      {"B1", "TC", 200},
266      {"D3", "D7", 100},
267      {"D3", "ThuVien", 40},
268      {"ThuVien", "D7", 20},
269  };
270  int so_canh = sizeof(ds_canh) / sizeof(Canh);
271
272  int lay_khoang_cach(const char *nguon, const char *dich) {
273      for (int i = 0; i < so_canh; i++) {
274          if ((strcmp(ds_canh[i].toa1, nguon) == 0 && strcmp(ds_canh[i].toa2, dich) == 0) ||
275              (strcmp(ds_canh[i].toa2, nguon) == 0 && strcmp(ds_canh[i].toa1, dich) == 0)) {
276              return ds_canh[i].khoang_cach;
277          }
278      }
279      return -1; // Khong tim thay
280  }
```

+ Struct Canh lưu thông tin mỗi cạnh: 2 đỉnh (toa1, toa2) và khoảng cách giữa chúng (khoang_cach).
+ Mảng ds_canh[] chứa các cạnh đã định nghĩa sẵn.
+ Hàm lay_khoang_cach nhận 2 tên đỉnh (nguon, dich), tìm trong mảng cạnh xem có cạnh nào nối 2 đỉnh này (theo cả 2 chiều vì đồ thị vô hướng).
+ Nếu tìm thấy, trả về khoảng cách; nếu không trả về -1.

Quay lại chương trình làm việc

Code chương trình

21. Thêm hàm để không lỗi khi gọi

```
282  // Thêm hàm timIndex để không lỗi khi gọi
283  int timIndex(char* tenToaNha[], int soDinh, const char* toa) {
284      for (int i = 0; i < soDinh; i++) {
285          if (strcmp(tenToaNha[i], toa) == 0) {
286              return i;
287          }
288      }
289      return -1; // Khong tim thay
290  }
```

+ Nhận vào:
  - Mảng chuỗi tenToaNha[] chứa tên các tòa nhà.
  - Số lượng tòa nhà soDinh.
  - Chuỗi toa là tên tòa nhà cần tìm.

+ Duyệt từ đầu đến cuối mảng:
  - So sánh từng phần tử với toa bằng strcmp.
  - Nếu bằng, trả về vị trí (index) của phần tử đó.
+ Nếu không tìm thấy, trả về -1.

Quay lại chương trình làm việc

Code chương trình

22. Hàm main

```c
int main() {
    char *tenToaNha[] = {
        "D3", "D3-5", "D5", "D7", "D9", "B1", "C7", "C2", "TC", "B3", "ThuVien"
    };
    int soDinh = sizeof(tenToaNha) / sizeof(tenToaNha[0]);

    Dothi *doThi = createDothi(soDinh);
    char nguon[20], dich[20];
    int lc1;


    while(1){
        printf("---- Duong di ngan nhat -------\n");
        printf("1. Nhap vi tri cac toa\n");
        printf("0. Ket thuc chuong trinh\n");
        printf("Chon: ");
        scanf("%d", &lc1);

        if(lc1 == 1 ){
            // Thêm các cạnh vào đồ thị dựa trên ds_canh
            for (int i = 0; i < so_canh; i++) {
                int u = timIndex(tenToaNha, soDinh, ds_canh[i].toa1);
                int v = timIndex(tenToaNha, soDinh, ds_canh[i].toa2);
                int trong_so = ds_canh[i].khoang_cach;

                if (u != -1 && v != -1) {
                    them_canh(doThi, u, v, trong_so);
                }
            }

            printf("Nhap ten dinh nguon: ");
            scanf("%s", nguon);
            printf("Nhap ten dinh dich: ");
            scanf("%s", dich);
```

```c
            printf("Nhap ten dinh nguon: ");
            scanf("%s", nguon);
            printf("Nhap ten dinh dich: ");
            scanf("%s", dich);

            int dinhNguon = timIndex(tenToaNha, soDinh, nguon);
            int dinhDich = timIndex(tenToaNha, soDinh, dich);

            if (dinhNguon == -1 || dinhDich == -1) {
                printf("Khong tim thay dinh nguon hoac dinh dich!\n");
                giaiPhongDoThi(doThi);
                return 1;
            }

            int *khoangCach = (int *)malloc(soDinh * sizeof(int));
            int *parent = (int *)malloc(soDinh * sizeof(int));

            dijkstra(doThi, dinhNguon, khoangCach, parent);

            if (khoangCach[dinhDich] == INT_MAX) {
                printf("Khong co duong di tu %s den %s\n", nguon, dich);
            } else {
                printf("Duong di ngan nhat tu %s den %s la: ", nguon, dich);
                inDuongDi(parent, dinhDich, tenToaNha);
                printf("\nTong khoang cach: %d met\n", khoangCach[dinhDich]);
                if(strcmp(nguon,"TC") == 0 && strcmp(dich,"D3") == 0){
                    system("start TCdenD3.png");
                }
```

Quay lại chương trình làm việc