**Permasalahan**

Seorang kurir harus pergi dari satu rumah ke rumah lainnya dimana rumah tersebut diberi nama , , , dan . Diketahui jarak antar rumah :

Titik awal kurir ke rumah terlebih dahulu. Buatlah algoritma untuk mencari solusi dari permasalahan tersebut!

**Penyelesaian**

Untuk mencari solusi dari permasalahan tersebut yaitu mencari rute terpendek untuk mengunjungi setiap rumah, perlu dilakukan pembuatan model terlebih dahulu untuk nanti dapat diproses menggunakan algoritma rekursif . Model tersebut berupa matriks yang merepresentasikan hubungan setiap rumah beserta jaraknya.



Algoritma akan membagi permasalahan tersebut menjadi upa-masalah yang lebih kecil dan setelah terbagi menjadi satuan yang lebih kecil, algoritma akan langsung mencari solusi dari upa-masalah tersebut. Pada persalahan di atas, setiap rumah memiliki rute termasuk rute menuju rumah itu sendiri. Algoritma harus mencari rute terpendek menuju rumah lain tetapi bukan menuju rumah itu sendiri dan bukan menuju rumah yang telah dikunjungi.

Sebagai contoh titik awal kurir itu ke rumah terlebih dahulu. Pada rumah terdapat rute termasuk rute menuju rumah itu sendiri yaitu . Kurir tersebut pasti akan memilih rute menuju rumah karena rute tersebut adalah rute terpendek dari semua rute yang ada. Kurir tidak akan memilih rute menuju rumah yang telah dia kunjungi. Jika menggunakan algoritma dapat diilustrasikan sebagai berikut.



Dengan algoritma , permasalahan akan dibagi menjadi beberapa upa-masalah sampai upa-masalah tersebut masing-masing memiliki rute untuk nanti dibandingkan mana rute terkecil pada masing-masing upa-masalah. Pada ilustrasi di atas, Permasalah dibagi menjadi upa-masalah dan . Solusi dari upa-masalah adalah tetapi solusi dari upa-masalah adalah karena rute dengan jarak merupakan rute menuju rumah itu sendiri dan rumah itu sudah dikunjungi karena menjadi titik awal kurir. Dan solusi dari permasalahan adalah rute dengan jarak menuju rumah .

Algoritma akan mengulangi proses pencarian rute terpendek tersebut dengan membagi menjadi upa-masalah kecil sedemikian rupa sehingga tidak ada lagi rumah yang belum dikunjungi oleh kurir.

**Pseudo-code**

Pada penjelasan di atas dapat dibuat suatu pseudo-code untuk algoritma dalam mencari rute terpendek sebagai berikut.

|  |
| --- |
| ***procedure*** *CariRuteTerpendek***(*in*** *graph* : ***Matrix***, *rumah*, *start*, *end* : ***int***, ***out*** *edge*, *vertex* : ***int*)**  ***deklarasi***  *mid* : ***int***  *e*   : ***Static*** ***Array*(*int***, 2**)**  *v*   : ***Static*** ***Array*(*int***, 2**)**  *s*   : ***Array*(*int***, 2**)**  ***algoritma***  *if* *end* - *start* == 1 *then*  *s***[**0**]** *graph***[***rumah***][***start***]**  *s***[**1**]** *graph***[***rumah***][***end***]**  *if* *s***[**0**]** < *s***[**1**]** *then*  *if* *s***[**0**]** != 0 *and* *s***[**0**]** != *edge* *then*  *vertex* *start*  *edge* *s***[**0**]**  *else*  *vertex* *end*  *edge* *s***[**1**]**  *endif*  *else*  *if* *s***[**1**]** != 0 *and* *s***[**1**]** != *edge* *then*  *vertex* *end*  *edge* *s***[**1**]**  *else*  *vertex* *start*  *edge* *s***[**0**]**  *endif*  *endif*  *else*  *mid* **(***start* + *end***)** *div* 2  *CariRuteTerpendek***(***graph*, *rumah*, *start*, *mid*, *e***[**0**]**, *v***[**0**])**  *CariRuteTerpendek***(***graph*, *rumah*, *mid* + 1, *end*, *e***[**1**]**, *v***[**1**])**  *if* *e***[**0**]** < *e***[**1**]** *then*  *if* **{***v***[**0**]}** *Solusi* == *then*                      vertex *v***[**0**]**; *v***[**1**]** *v***[**0**]**                      edge *e***[**0**]**; *e***[**1**]** *e***[**0**]**  *else*  *vertex* *v***[**1**]**; *v***[**0**]** *v***[**1**]**  *edge* *e***[**1**]**; *e***[**0**]** *e***[**1**]**  *endif*  *else*  *if* **{***v***[**1**]}** *Solusi* == *then*  *vertex* *v***[**1**]**; *v***[**0**]** *v***[**1**]** |
| *edge* *e***[**1**]**; *e***[**0**]** *e***[**1**]**  *else*  *vertex* *v***[**0**]**; *v***[**1**]** *v***[**0**]**  *edge* *e***[**0**]**; *e***[**1**]** *e***[**0**]**  *endif*  *endif*  *if* **{***vertex***}** *Solusi* != *then*  *Solusi* *Solusi* **{(***rumah*, *graph***[***rumah***][**3**])}**  *Solusi* *Solusi* **{(**3, 0**)}**  *else*  *Solusi* *Solusi* **{(***rumah*, *edge***)}**  *CariRuteTerpendek***(***vertex*, 0, 3, *edge*, *vertex***)**  *endif*  *endif* |

**Pembuktian**

Untuk membuktikan kebenaran dari algoritma yang telah dibuat pada bagian atas. Saya telah membuat implementasi program dengan menggunakan bahasa pemrograman ***C++*** sebagai berikut.

|  |
| --- |
| #include <iostream>  #include <vector>  int graph[4][4] = {      {0, 30, 28, 41}, */\* Rumah A \*/*      {30, 0, 18, 15}, */\* Rumah B \*/*      {28, 18, 0, 21}, */\* Rumah C \*/*      {41, 15, 21, 0}, */\* Rumah D \*/*  };  std::vector<std::pair<int, int>> Solusi;  bool isVisited(const int& vertex) {      for (const std::pair<int, int>& i : Solusi) {          if (vertex == i.first) {              return true;          }      }      return false;  } |
| void ShortestRoute(int rumah, int start, int end, int& edge, int& vertex) {  */\* deklarasi \*/*      int mid;      static int e[2];      static int v[2];      int s[2];  */\* algoritma \*/*      if (end - start == 1) {          s[0] = graph[rumah][start];          s[1] = graph[rumah][end];          if (s[0] < s[1]) {              if (s[0] != 0 and s[0] != edge) {                  vertex = start;                  edge = s[0];              } else {                  vertex = end;                  edge = s[1];              }          } else {              if (s[1] != 0 and s[1] != edge) {                  vertex = end;                  edge = s[1];              } else {                  vertex = start;                  edge = s[0];              }          }      } else {          mid = (start + end) / 2;          ShortestRoute(rumah, start, mid, e[0], v[0]);          ShortestRoute(rumah, mid + 1, end, e[1], v[1]);          if (e[0] < e[1]) {              if (not isVisited(v[0])) {                  vertex = v[1] = v[0];                  edge = e[1] = e[0];              } else {                  vertex = v[0] = v[1];                  edge = e[0] = e[1];              }          } else {              if (not isVisited(v[1])) {                  vertex = v[0] = v[1]; |
| edge = e[0] = e[1];              } else {                  vertex = v[1] = v[0];                  edge = e[1] = e[0];              }          }          if (isVisited(vertex)) {              Solusi.push\_back({rumah, graph[rumah][3]});              Solusi.push\_back({3, 0});          } else {              Solusi.push\_back({rumah, edge});              ShortestRoute(vertex, 0, 3, edge, vertex);          }      }  }  char Change(int index) {      switch (index) {          case 0: return 'A';          case 1: return 'B';          case 2: return 'C';          case 3: return 'D';      }      return ' ';  }  int main() {      static int edge, vertex;      ShortestRoute(3, 0, 3, edge, vertex);      int sum = Solusi.at(1).second;      for (int i = 0; i < Solusi.size() - 1; i++) {          std::cout << "Kurir pergi dari rumah " << Change(Solusi.at(i).first)                    << " ke rumah " << Change(Solusi.at(i + 1).first)                    << " dengan jarak " << Solusi.at(i).second << "m\n";          sum += Solusi.at(i + 1).second;      }      std::cout << "Total perjalanan kurir : " << sum << "m\n";      return 0;  } |

|  |
| --- |
|  |