SOAL 1 : Anda memiliki kumpulan koin dengan nilai 1, 3, dan 4. Bagaimana Anda menggunakan algoritma Greedy untuk menentukan jumlah minimum koin yang diperlukan untuk membentuk jumlah tertentu, misalnya 6?\*

Jawaban Anda

Dalam konteks algoritma Greedy, masalah ini dapat diformulasikan sebagai berikut:

1. **Himpunan Kandidat (C)**: Ini adalah kumpulan koin dengan nilai 1, 3, dan 4. Kita bisa menggunakan koin ini untuk membentuk jumlah yang diinginkan.
2. **Himpunan Solusi (S)**: Ini adalah kumpulan koin yang telah kita pilih. Awalnya, himpunan ini kosong. Dalam konteks matematis, kita bisa menggambarkannya sebagai vektor X = {x1, x2, ..., xn}, di mana xi = 1 jika koin di dipilih dan xi = 0 jika koin di tidak dipilih.
3. **Fungsi Solusi**: Fungsi ini memeriksa apakah total nilai koin yang telah kita pilih sama dengan jumlah yang kita inginkan. Dalam kasus ini, kita ingin membentuk jumlah 6. Dalam notasi matematika, kita bisa menulis ini sebagai ∑di\*xi = A, di mana A adalah jumlah yang kita inginkan dan di adalah nilai koin.
4. **Fungsi Seleksi**: Fungsi ini memilih koin dengan nilai tertinggi dari himpunan koin yang tersisa. Dengan kata lain, kita selalu mencoba menggunakan koin dengan nilai tertinggi yang memungkinkan pada setiap langkah.
5. **Fungsi Kelayakan**: Fungsi ini memeriksa apakah koin yang baru kita pilih, jika ditambahkan ke total koin yang telah kita pilih sebelumnya, tidak akan melebihi jumlah yang kita inginkan. Dalam hal ini, jumlah yang kita inginkan adalah 6.
6. **Fungsi Objektif**: Fungsi ini mencoba meminimalkan jumlah koin yang kita gunakan. Dalam kata lain, kita ingin mencapai jumlah yang diinginkan dengan menggunakan sebanyak mungkin koin berharga tinggi. Dalam notasi matematika, kita bisa menulis ini sebagai Minimize F = ∑xi, yang berarti kita mencoba meminimalkan jumlah koin yang kita gunakan.

Penyelesaian dengan pendekatan exhaustive search akan memeriksa semua kemungkinan solusi. Karena setiap koin bisa dipilih atau tidak, ada 2^n kemungkinan solusi, di mana n adalah jumlah koin. Untuk setiap solusi, kita perlu mengevaluasi fungsi objektif, yang membutuhkan waktu O(n). Oleh karena itu, kompleksitas waktu total algoritma exhaustive search adalah O(n \* 2^n), yang bisa sangat besar untuk n yang besar.

Namun, dengan menggunakan algoritma Greedy, kita dapat menemukan solusi dengan kompleksitas waktu yang jauh lebih rendah. Dalam hal ini, kita selalu memilih koin dengan nilai tertinggi yang memungkinkan pada setiap langkah, yang memastikan bahwa kita selalu membuat pilihan yang optimal secara lokal pada setiap langkah dalam upaya untuk menemukan solusi yang optimal secara global.

Berikut adalah langkah-langkah yang kita ambil untuk menyelesaikan masalah ini:

1. Urutkan koin dari nilai tertinggi ke terendah. Dalam kasus ini, urutannya adalah 4, 3, 1.
2. Pilih koin dengan nilai tertinggi yang tidak melebihi jumlah yang kita inginkan. Dalam kasus ini, kita memilih koin 4. Sekarang, total kita adalah 4, dan kita masih perlu menemukan cara untuk membentuk jumlah 2 lagi.
3. Dari sisa jumlah (2), kita tidak bisa memilih koin 3 karena nilai ini melebihi jumlah yang kita inginkan. Oleh karena itu, kita memilih koin 1. Sekarang, total kita adalah 5, dan kita masih perlu menemukan cara untuk membentuk jumlah 1 lagi.
4. Dari sisa jumlah (1), kita memilih koin 1 lagi. Sekarang, total kita adalah 6, yang merupakan jumlah yang kita inginkan.

Oleh karena itu, jumlah minimum koin yang kita butuhkan untuk membentuk jumlah 6 adalah 3 koin, yaitu koin 4, koin 1, dan koin 1.

Sumber: Munir, R. (2021). Algoritma Greedy. Dalam Program Studi Teknik Informatika STEI ITB (hlm. 3-13, 17-19). Bagian 1.

SOAL 2 : Anda memiliki daftar kegiatan dengan waktu mulai dan waktu selesai masing-masing. Bagaimana Anda menggunakan algoritma Greedy (misalnya, algoritma Jadwal Pekerjaan yang Kompatibel) untuk menemukan jumlah maksimum kegiatan yang dapat Anda jalani tanpa saling tumpang tindih?\*

Jawaban Anda

Dalam konteks algoritma Greedy, masalah ini dapat diformulasikan sebagai berikut:

1. **Himpunan Kandidat (C)**: Ini adalah daftar semua kegiatan yang tersedia. Setiap kegiatan diwakili oleh waktu mulai dan waktu selesai.
2. **Himpunan Solusi (S)**: Ini adalah daftar kegiatan yang telah kita pilih untuk dijalani. Awalnya, himpunan ini kosong.
3. **Fungsi Solusi**: Fungsi ini menentukan apakah daftar kegiatan yang telah kita pilih tidak saling tumpang tindih. Dalam hal ini, solusi adalah daftar kegiatan di mana tidak ada dua kegiatan yang tumpang tindih.
4. **Fungsi Seleksi**: Fungsi ini memilih kegiatan berikutnya yang akan ditambahkan ke dalam himpunan solusi. Dalam kasus ini, strategi Greedy yang kita gunakan adalah memilih kegiatan yang berakhir paling awal. Dengan memilih kegiatan yang berakhir paling awal, kita memberi diri kita kesempatan terbesar untuk menghadiri kegiatan lain setelahnya.
5. **Fungsi Kelayakan**: Fungsi ini memeriksa apakah kegiatan yang dipilih dapat ditambahkan ke dalam himpunan solusi. Dalam konteks ini, kegiatan adalah layak jika dan hanya jika waktu mulai kegiatan tersebut tidak tumpang tindih dengan waktu selesai kegiatan lain yang sudah ada di dalam himpunan solusi.
6. **Fungsi Objektif**: Fungsi ini mencoba memaksimalkan jumlah kegiatan yang dapat kita jalani. Dalam konteks ini, kita mencoba memaksimalkan jumlah kegiatan dalam himpunan solusi kita.

Untuk menerapkan algoritma ini:

1. Urutkan semua kegiatan berdasarkan waktu selesai mereka.
2. Pilih kegiatan dengan waktu selesai terawal dan tambahkan ke dalam himpunan solusi.
3. Dari kegiatan yang tersisa, pilih kegiatan dengan waktu mulai yang tidak tumpang tindih dengan kegiatan yang sudah ada di dalam himpunan solusi dan memiliki waktu selesai terawal. Ulangi langkah ini sampai tidak ada lagi kegiatan yang bisa ditambahkan ke dalam himpunan solusi.

Dalam konteks penjadwalan pekerjaan, kita juga bisa mempertimbangkan profit atau keuntungan yang diperoleh dari setiap pekerjaan. Dalam hal ini, kita mungkin ingin memaksimalkan total keuntungan dari pekerjaan yang kita pilih, yang bisa menjadi fungsi objektif kita. Dalam hal ini, kita mungkin perlu mengubah fungsi seleksi kita untuk memilih pekerjaan dengan profit tertinggi yang bisa kita lakukan tanpa tumpang tindih dengan pekerjaan yang sudah kita pilih.

Pendekatan exhaustive search akan memeriksa semua kemungkinan kombinasi pekerjaan dan memilih kombinasi dengan total keuntungan terbesar yang tidak saling tumpang tindih. Namun, pendekatan ini bisa sangat tidak efisien, karena ada 2^n kemungkinan kombinasi pekerjaan, di mana n adalah jumlah pekerjaan. Dengan menggunakan algoritma Greedy, kita bisa menemukan solusi yang cukup baik dengan cara yang lebih efisien. Dalam hal ini, kita selalu memilih pekerjaan berikutnya dengan waktu selesai terawal atau profit tertinggi (tergantung pada fungsi objektif kita) yang bisa kita lakukan tanpa tumpang tindih dengan pekerjaan yang sudah kita pilih.

Sumber: Munir, R. (2021). Algoritma Greedy. Dalam Program Studi Teknik Informatika STEI ITB (hlm. 58-67). Bagian 1.

SOAL 3 : Anda memiliki graf berbobot positif yang mewakili jaringan komunikasi antar kota. Bagaimana Anda menggunakan algoritma Greedy (misalnya, algoritma Minimum Spanning Tree) untuk menemukan jalur komunikasi dengan total bobot minimum?\*

Jawaban Anda

Dalam konteks algoritma Greedy, masalah ini dapat diformulasikan sebagai berikut:

1. **Himpunan Kandidat (C)**: Ini adalah semua sisi dalam graf, di mana setiap sisi mewakili jalur komunikasi antara dua kota dan bobotnya mewakili biaya komunikasi.
2. **Himpunan Solusi (S)**: Ini adalah sisi-sisi yang telah kita pilih untuk membentuk Pohon Merentang Minimum (MST). Awalnya, himpunan ini kosong.
3. **Fungsi Solusi**: Fungsi ini menentukan apakah himpunan sisi yang telah kita pilih membentuk pohon penjangkauan. Pohon penjangkauan adalah pohon yang mencakup semua simpul dalam graf.
4. **Fungsi Seleksi**: Fungsi ini memilih sisi dengan bobot terendah dari himpunan sisi yang tersisa.
5. **Fungsi Kelayakan**: Fungsi ini memeriksa apakah sisi yang baru kita pilih, jika ditambahkan ke MST yang telah kita bentuk sejauh ini, tidak akan membentuk siklus.
6. **Fungsi Objektif**: Fungsi ini mencoba meminimalkan total bobot dari sisi-sisi dalam MST.

Berikut adalah langkah-langkah yang kita ambil untuk menyelesaikan masalah ini menggunakan algoritma Prim dan Kruskal:

**Algoritma Prim**:  
Algoritma ini memulai dengan memilih sisi dengan bobot terendah dan secara bertahap menambahkan sisi dengan bobot terendah yang terhubung dengan simpul yang telah dipilih, asalkan sisi baru tersebut tidak membentuk siklus. Proses ini diulangi sampai pohon merentang minimum mencakup semua simpul dalam graf.

1. Cari sisi dengan bobot terkecil dalam graf dan tambahkan ke dalam MST.
2. Cari sisi dengan bobot terkecil yang terhubung dengan simpul yang ada dalam MST dan tidak membentuk siklus jika ditambahkan. Tambahkan sisi tersebut ke dalam MST.
3. Ulangi langkah 2 sampai semua simpul termasuk dalam MST.

Kompleksitas waktu algoritma Prim adalah O(n^2), di mana n adalah jumlah simpul dalam graf.

**Algoritma Kruskal**:  
Algoritma ini memulai dengan mengurutkan semua sisi berdasarkan bobotnya dari yang terkecil hingga terbesar. Kemudian, sisi dengan bobot terkecil ditambahkan ke dalam MST asalkan tidak membentuk siklus. Proses ini diulangi sampai MST mencakup semua simpul dalam graf.

1. Urutkan semua sisi berdasarkan bobotnya dari yang terkecil hingga terbesar.
2. Pilih sisi dengan bobot terkecil dan tambahkan ke dalam MST, asalkan tidak membentuk siklus.
3. Ulangi langkah 2 sampai semua simpul termasuk dalam MST.

Kompleksitas waktu algoritma Kruskal adalah O(E log E), di mana E adalah jumlah sisi dalam graf.

Kedua algoritma ini menghasilkan pohon merentang minimum, yang mewakili jalur komunikasi antar kota dengan total bobot minimum. Namun, pohon merentang minimum yang dihasilkan oleh algoritma Prim dan Kruskal mungkin berbeda jika ada lebih dari satu pohon merentang minimum.

Sumber: Munir, R. (2021). Algoritma Greedy. Dalam Program Studi Teknik Informatika STEI ITB (hlm. 2-12). Bagian 2.

SOAL 4 : Anda memiliki himpunan interval, masing-masing merepresentasikan waktu mulai dan waktu selesai suatu kegiatan. Bagaimana Anda menggunakan algoritma Greedy untuk menemukan jumlah maksimum interval yang tidak tumpang tindih?

Jawaban Anda

Dalam konteks algoritma Greedy, masalah ini dapat diformulasikan sebagai berikut:

1. **Himpunan Kandidat (C)**: Ini adalah semua interval atau aktivitas yang tersedia. Setiap aktivitas diwakili oleh waktu mulai dan waktu selesai.
2. **Himpunan Solusi (S)**: Ini adalah daftar aktivitas yang telah kita pilih untuk dijalankan. Awalnya, himpunan ini kosong.
3. **Fungsi Solusi**: Fungsi ini menentukan apakah daftar aktivitas yang telah kita pilih tidak saling tumpang tindih. Dalam hal ini, solusi adalah daftar kegiatan di mana tidak ada dua kegiatan yang tumpang tindih.
4. **Fungsi Seleksi**: Fungsi ini memilih kegiatan berikutnya yang akan ditambahkan ke dalam himpunan solusi. Dalam kasus ini, strategi Greedy yang kita gunakan adalah memilih kegiatan yang berakhir paling awal. Dengan memilih kegiatan yang berakhir paling awal, kita memberi diri kita kesempatan terbesar untuk menghadiri kegiatan lain setelahnya.
5. **Fungsi Kelayakan**: Fungsi ini memeriksa apakah kegiatan yang dipilih dapat ditambahkan ke dalam himpunan solusi. Dalam konteks ini, kegiatan adalah layak jika dan hanya jika waktu mulai kegiatan tersebut lebih besar atau sama dengan waktu selesai kegiatan lain yang sudah ada di dalam himpunan solusi.
6. **Fungsi Objektif**: Fungsi ini mencoba memaksimalkan jumlah kegiatan yang dapat kita jalani. Dalam konteks ini, kita mencoba memaksimalkan jumlah kegiatan dalam himpunan solusi kita.

Untuk menerapkan algoritma ini:

1. Urutkan semua kegiatan berdasarkan waktu selesai mereka dari yang paling awal hingga yang paling akhir.
2. Pilih kegiatan dengan waktu selesai terawal dan tambahkan ke dalam himpunan solusi.
3. Dari kegiatan yang tersisa, pilih kegiatan dengan waktu mulai yang lebih besar atau sama dengan waktu selesai kegiatan terakhir yang ditambahkan ke dalam himpunan solusi. Ulangi langkah ini sampai tidak ada lagi kegiatan yang bisa ditambahkan ke dalam himpunan solusi.

Algoritma ini memiliki kompleksitas waktu O(n) jika waktu pengurutan tidak diperhitungkan. Selain itu, kita dapat membuktikan bahwa algoritma ini menghasilkan solusi optimal. Misalkan S = {1, 2,…, n} adalah himpunan aktivitas yang sudah diurutkan berdasarkan waktu selesai. Asumsikan A ⊆ S adalah solusi optimal. Jika aktivitas pertama dalam A adalah 1, maka A dimulai dengan pilihan Greedy. Jika bukan, kita dapat menunjukkan bahwa ada solusi optimal lainnya B = (A - {k1}) ∪ {1}, yang dimulai dengan pilihan Greedy (aktivitas 1). Karena f1 ≤ fk1, dan aktivitas dalam A kompatibel, aktivitas dalam B juga kompatibel. Karena |A| = |B|, maka B juga optimal. Oleh karena itu, selalu ada solusi optimal yang dimulai dengan pilihan Greedy

Sumber: Munir, R. (2021). Algoritma Greedy. Dalam Program Studi Teknik Informatika STEI ITB (hlm. 21-31). Bagian 1.

SOAL 5 : Anda memiliki sekumpulan barang dengan harga dan berat masing-masing. Bagaimana Anda menggunakan algoritma Greedy (misalnya, algoritma Knapsack Fractional) untuk memilih barang-barang yang dapat dimasukkan ke dalam ransel dengan kapasitas tertentu sehingga nilai totalnya maksimal?\*

Jawaban Anda

Dalam konteks algoritma Greedy, masalah ini dapat diformulasikan sebagai berikut:

1. **Himpunan Kandidat (C)**: Ini adalah semua barang yang tersedia. Setiap barang diwakili oleh berat dan harga.
2. **Himpunan Solusi (S)**: Ini adalah daftar barang yang telah kita pilih untuk dimasukkan ke dalam ransel. Awalnya, himpunan ini kosong.
3. **Fungsi Solusi**: Fungsi ini menentukan apakah daftar barang yang telah kita pilih tidak melebihi kapasitas ransel. Dalam hal ini, solusi adalah daftar barang di mana total berat barang tidak melebihi kapasitas ransel.
4. **Fungsi Seleksi**: Fungsi ini memilih barang berikutnya yang akan ditambahkan ke dalam himpunan solusi. Dalam kasus ini, strategi Greedy yang kita gunakan adalah memilih barang dengan rasio harga per berat (pi/wi) tertinggi. Dengan memilih barang dengan rasio harga per berat tertinggi, kita berusaha memaksimalkan nilai total barang yang kita masukkan ke dalam ransel.
5. **Fungsi Kelayakan**: Fungsi ini memeriksa apakah barang yang dipilih dapat ditambahkan ke dalam himpunan solusi. Dalam konteks ini, barang adalah layak jika dan hanya jika total berat barang dalam himpunan solusi ditambah berat barang yang dipilih tidak melebihi kapasitas ransel. Jika barang tidak muat secara utuh, kita bisa memasukkan sebagian dari barang tersebut ke dalam ransel.
6. **Fungsi Objektif**: Fungsi ini mencoba memaksimalkan nilai total barang yang kita masukkan ke dalam ransel. Dalam konteks ini, kita mencoba memaksimalkan total harga barang dalam himpunan solusi kita.

Untuk menerapkan algoritma ini:

1. Hitung rasio harga per berat untuk setiap barang.
2. Urutkan semua barang berdasarkan rasio harga per berat mereka dari yang tertinggi ke yang terendah.
3. Mulai dari barang dengan rasio harga per berat tertinggi, coba masukkan barang tersebut ke dalam ransel. Jika barang tersebut muat secara utuh, masukkan seluruh barang ke dalam ransel. Jika tidak, masukkan sebagian dari barang tersebut ke dalam ransel sedemikian rupa sehingga total berat barang dalam ransel sama dengan kapasitas ransel.
4. Ulangi langkah 3 untuk barang-barang selanjutnya sampai ransel penuh atau tidak ada lagi barang yang bisa ditambahkan ke dalam ransel.

Algoritma ini memiliki kompleksitas waktu O(n) jika waktu pengurutan tidak diperhitungkan. Selain itu, kita dapat membuktikan bahwa algoritma ini menghasilkan solusi optimal jika barang dapat dibagi menjadi fraksi. Hal ini karena kita selalu memilih barang dengan rasio harga per berat tertinggi, yang memaksimalkan nilai total barang yang kita masukkan ke dalam ransel. Teorema 2 menunjukkan bahwa jika barang-barang sudah diurutkan berdasarkan rasio harga per berat yang menurun, algoritma Greedy dengan strategi pemilihan objek berdasarkan rasio harga per berat tertinggi akan menghasilkan solusi yang optimal.

Sumber: Munir, R. (2021). Algoritma Greedy. Dalam Program Studi Teknik Informatika STEI ITB (hlm. 50-57). Bagian 1.

SOAL 6 : Anda memiliki himpunan pekerjaan dengan waktu mulai, waktu selesai, dan profit masing-masing. Bagaimana Anda menggunakan algoritma Greedy (misalnya, algoritma Scheduling Pekerjaan Greedy) untuk memaksimalkan profit total?\*

Jawaban Anda

Dalam konteks algoritma Greedy, masalah ini dapat diformulasikan sebagai berikut:

1. **Himpunan Kandidat (C)**: Ini adalah semua pekerjaan yang tersedia. Setiap pekerjaan diwakili oleh waktu mulai, waktu selesai, dan profit.
2. **Himpunan Solusi (S)**: Ini adalah daftar pekerjaan yang telah kita pilih untuk dikerjakan. Awalnya, himpunan ini kosong.
3. **Fungsi Solusi**: Fungsi ini menentukan apakah daftar pekerjaan yang telah kita pilih tidak saling tumpang tindih dalam waktu dan semua pekerjaan selesai sebelum tenggat waktu mereka.
4. **Fungsi Seleksi**: Fungsi ini memilih pekerjaan berikutnya yang akan ditambahkan ke dalam himpunan solusi. Dalam kasus ini, strategi Greedy yang kita gunakan adalah memilih pekerjaan dengan profit tertinggi.
5. **Fungsi Kelayakan**: Fungsi ini memeriksa apakah pekerjaan yang dipilih dapat ditambahkan ke dalam himpunan solusi. Dalam konteks ini, pekerjaan layak jika dan hanya jika tidak ada pekerjaan lain dalam himpunan solusi yang tumpang tindih dalam waktu dengannya dan pekerjaan tersebut dapat diselesaikan sebelum tenggat waktunya.
6. **Fungsi Objektif**: Fungsi ini mencoba memaksimalkan total profit dari pekerjaan yang kita pilih.

Untuk menerapkan algoritma ini:

1. Urutkan semua pekerjaan berdasarkan profit mereka dalam urutan menurun.
2. Mulai dari pekerjaan dengan profit tertinggi, coba masukkan pekerjaan tersebut ke dalam himpunan solusi jika pekerjaan tersebut layak (yaitu, tidak ada pekerjaan lain dalam himpunan solusi yang tumpang tindih dalam waktu dengannya dan pekerjaan tersebut dapat diselesaikan sebelum tenggat waktunya).
3. Ulangi langkah 2 untuk pekerjaan-pekerjaan selanjutnya sampai semua pekerjaan telah dipertimbangkan.

Algoritma ini memiliki kompleksitas waktu O(n^2) jika waktu pengurutan tidak diperhitungkan, di mana n adalah jumlah pekerjaan. Selain itu, kita dapat membuktikan bahwa algoritma ini menghasilkan solusi yang optimal. Hal ini karena kita selalu memilih pekerjaan dengan profit tertinggi yang layak, yang memaksimalkan total profit dari pekerjaan yang kita pilih.

Sumber: Munir, R. (2021). Algoritma Greedy. Dalam Program Studi Teknik Informatika STEI ITB (hlm. 58-67). Bagian 1.

SOAL 7 : Anda memiliki graf berbobot positif yang mewakili jaringan komunikasi antar kota. Bagaimana Anda menggunakan algoritma Greedy (misalnya, algoritma Dijkstra) untuk menemukan jalur terpendek dari satu kota ke kota lain?\*

Jawaban Anda

Dalam konteks algoritma Greedy, masalah ini dapat diformulasikan sebagai berikut:

1. **Himpunan Kandidat (C)**: Ini adalah semua simpul dalam graf yang belum dikunjungi.
2. **Himpunan Solusi (S)**: Ini adalah daftar simpul yang telah dikunjungi. Awalnya, himpunan ini hanya berisi simpul awal.
3. **Fungsi Solusi**: Fungsi ini menentukan apakah kita telah mencapai simpul tujuan. Jika ya, kita telah menemukan jalur terpendek dari simpul awal ke simpul tujuan.
4. **Fungsi Seleksi**: Fungsi ini memilih simpul berikutnya yang akan dikunjungi. Dalam kasus ini, strategi Greedy yang kita gunakan adalah memilih simpul yang belum dikunjungi dengan jarak terpendek dari simpul awal.
5. **Fungsi Kelayakan**: Fungsi ini memeriksa apakah simpul yang dipilih dapat ditambahkan ke dalam himpunan solusi. Dalam konteks ini, simpul adalah layak jika dan hanya jika itu belum dikunjungi.
6. **Fungsi Objektif**: Fungsi ini mencoba meminimalkan total jarak dari simpul awal ke simpul tujuan.

Untuk menerapkan algoritma Dijkstra secara efisien, kita menggunakan struktur data priority queue atau min-heap untuk menyimpan simpul-simpul dengan jarak terpendek sementara dari simpul awal. Ini memungkinkan kita untuk menemukan simpul dengan jarak terpendek dalam waktu O(log n) pada setiap iterasi, daripada O(n) untuk pencarian linier.

Langkah-langkah algoritma Dijkstra dengan menggunakan priority queue atau min-heap adalah sebagai berikut:

1. Setel jarak untuk simpul awal menjadi 0 dan jarak untuk semua simpul lain menjadi tak hingga (∞). Masukkan simpul awal ke dalam priority queue.
2. Ambil simpul dengan jarak terpendek sementara dari priority queue. Ini adalah simpul yang akan dikunjungi berikutnya.
3. Untuk setiap simpul yang bersebelahan dengan simpul yang baru saja diambil dari priority queue, hitung jarak dari simpul awal melalui simpul tersebut. Jika jarak baru ini lebih pendek dari jarak sebelumnya, perbarui jaraknya dan masukkan simpul tersebut ke dalam priority queue.
4. Ulangi langkah 2 dan 3 sampai simpul tujuan diambil dari priority queue atau priority queue kosong (yang berarti tidak ada jalur dari simpul awal ke simpul tujuan).

Dengan menggunakan priority queue atau min-heap, kompleksitas waktu algoritma Dijkstra menjadi O((V+E) log V), di mana V adalah jumlah simpul dan E adalah jumlah sisi dalam graf. Ini lebih efisien daripada kompleksitas O(V^2) jika kita tidak menggunakan struktur data tambahan.

Algoritma Dijkstra dapat digunakan untuk menemukan jalur terpendek dalam berbagai aplikasi, seperti dalam sistem navigasi dan jaringan komputer untuk menentukan rute tercepat antara dua titik.

Sumber: Munir, R. (2021). Algoritma Greedy. Dalam Program Studi Teknik Informatika STEI ITB (hlm. 13-30). Bagian 2.

SOAL 8 : Anda memiliki kumpulan interval waktu yang merepresentasikan waktu mulai dan waktu selesai suatu acara. Bagaimana Anda menggunakan algoritma Greedy (misalnya, algoritma Interval Scheduling) untuk menemukan jumlah maksimum acara yang bisa Anda hadiri tanpa tumpang tindih?\*

Jawaban Anda

Dalam konteks algoritma Greedy, masalah ini dapat diformulasikan sebagai berikut:

1. **Himpunan Kandidat (C)**: Ini adalah semua interval waktu atau acara yang tersedia. Setiap acara diwakili oleh waktu mulai dan waktu selesai.
2. **Himpunan Solusi (S)**: Ini adalah daftar acara yang telah kita pilih untuk hadiri. Awalnya, himpunan ini kosong.
3. **Fungsi Solusi**: Fungsi ini menentukan apakah daftar acara yang telah kita pilih tidak saling tumpang tindih dalam waktu.
4. **Fungsi Seleksi**: Fungsi ini memilih acara berikutnya yang akan ditambahkan ke dalam himpunan solusi. Dalam kasus ini, strategi Greedy yang kita gunakan adalah memilih acara yang berakhir paling awal.
5. **Fungsi Kelayakan**: Fungsi ini memeriksa apakah acara yang dipilih dapat ditambahkan ke dalam himpunan solusi. Dalam konteks ini, acara layak jika dan hanya jika tidak ada acara lain dalam himpunan solusi yang tumpang tindih dalam waktu dengannya.
6. **Fungsi Objektif**: Fungsi ini mencoba memaksimalkan jumlah acara yang bisa kita hadiri.

Untuk menerapkan algoritma Interval Scheduling:

1. Urutkan semua acara berdasarkan waktu selesai mereka dalam urutan non-menurun.
2. Mulai dari acara dengan waktu selesai paling awal, tambahkan acara tersebut ke dalam himpunan solusi jika acara tersebut layak (yaitu, tidak ada acara lain dalam himpunan solusi yang tumpang tindih dalam waktu dengannya).
3. Ulangi langkah 2 untuk acara-acara selanjutnya sampai semua acara telah dipertimbangkan.

Algoritma ini memiliki kompleksitas waktu O(n log n) jika waktu pengurutan diperhitungkan, di mana n adalah jumlah acara. Selain itu, kita dapat membuktikan bahwa algoritma ini menghasilkan solusi yang optimal. Hal ini karena kita selalu memilih acara yang berakhir paling awal dan layak, yang memaksimalkan jumlah acara yang bisa kita hadiri.

Sumber:

<https://www.geeksforgeeks.org/scheduling-in-greedy-algorithms/>

Princeton University. ????. Interval Scheduling. Dalam Chapter 4.4.1.

SOAL 9 : Anda memiliki himpunan pegunungan dengan ketinggian yang berbeda-beda. Bagaimana Anda menggunakan algoritma Greedy (misalnya, algoritma Hill Climbing) untuk menemukan jalur pendakian yang memaksimalkan ketinggian yang dinaiki?\*

Jawaban Anda

Dalam konteks algoritma Greedy, masalah ini dapat diformulasikan sebagai berikut:

1. **Himpunan Kandidat (C)**: Ini adalah semua pegunungan yang tersedia. Setiap pegunungan diwakili oleh ketinggiannya.
2. **Himpunan Solusi (S)**: Ini adalah jalur pendakian yang telah kita pilih. Awalnya, jalur ini kosong.
3. **Fungsi Solusi**: Fungsi ini menentukan apakah jalur pendakian yang telah kita pilih mencapai ketinggian maksimum. Jika tidak ada lagi pegunungan yang lebih tinggi dan dapat dijangkau, maka jalur pendakian tersebut adalah solusi maksimal yang dapat diperoleh dengan algoritma Hill Climbing.
4. **Fungsi Seleksi**: Fungsi ini memilih pegunungan berikutnya yang akan ditambahkan ke dalam jalur pendakian. Dalam kasus ini, strategi Greedy yang kita gunakan adalah memilih pegunungan dengan ketinggian tertinggi yang dapat dijangkau dari pegunungan saat ini.
5. **Fungsi Kelayakan**: Fungsi ini memeriksa apakah pegunungan yang dipilih dapat ditambahkan ke dalam jalur pendakian. Dalam konteks ini, pegunungan layak jika dan hanya jika ketinggiannya lebih tinggi dari pegunungan sebelumnya dalam jalur pendakian dan terhubung langsung (dapat dijangkau) dari pegunungan sebelumnya.
6. **Fungsi Objektif**: Fungsi ini mencoba memaksimalkan ketinggian total yang dinaiki dalam jalur pendakian.

Untuk menerapkan algoritma Hill Climbing:

1. Mulai dari pegunungan dengan ketinggian terendah sebagai titik awal.
2. Cari semua pegunungan yang ketinggiannya lebih tinggi dari pegunungan saat ini dan dapat dijangkau langsung dari pegunungan saat ini.
3. Dari semua pegunungan yang memenuhi syarat di langkah 2, pilih pegunungan dengan ketinggian tertinggi sebagai pegunungan berikutnya dalam jalur pendakian.
4. Ulangi langkah 2 dan 3 sampai tidak ada lagi pegunungan yang ketinggiannya lebih tinggi dan dapat dijangkau langsung dari pegunungan saat ini.

Algoritma Hill Climbing memiliki beberapa kelemahan:

1. Tidak selalu menghasilkan solusi optimal global, tetapi hanya solusi optimal lokal. Algoritma ini bisa terjebak di pegunungan lokal yang ketinggiannya lebih rendah dari pegunungan lain yang tidak dapat dijangkau langsung.
2. Kinerja algoritma sangat bergantung pada titik awal yang dipilih. Jika titik awal terlalu jauh dari solusi optimal global, algoritma akan kesulitan untuk mencapainya.
3. Kompleksitas waktu algoritma bergantung pada jumlah pegunungan dan bagaimana mereka tersebar. Dalam kasus terburuk, algoritma mungkin harus memeriksa semua pegunungan untuk menemukan solusi optimal lokal.

Untuk mengatasi kelemahan-kelemahan tersebut, dapat digunakan variasi dari algoritma Hill Climbing, seperti:

1. **Random Restart Hill Climbing**: Algoritma ini dijalankan berkali-kali dari titik awal yang berbeda-beda untuk meningkatkan peluang menemukan solusi optimal global.
2. **Simulated Annealing**: Algoritma ini mengizinkan kemunduran sementara (bergerak ke pegunungan yang lebih rendah) dengan probabilitas tertentu untuk mencegah terjebak di optimum lokal.
3. **Genetic Algorithm**: Algoritma ini menggunakan konsep evolusi dan mutasi untuk mencari solusi optimal global dengan cara menggabungkan dan memodifikasi solusi-solusi lokal yang ditemukan sebelumnya.

Dengan menggunakan variasi-variasi tersebut, algoritma Hill Climbing dapat menjadi lebih efektif dalam menemukan solusi optimal global untuk masalah-masalah optimasi yang kompleks.

Sumber:

<https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/hill-climbing-algorithm-in-ai>

<https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/>

SOAL 10 : Anda memiliki himpunan pegunungan dengan ketinggian yang berbeda-beda. Bagaimana Anda menggunakan algoritma Greedy (misalnya, algoritma Best-First Search) untuk menemukan jalur pendakian yang meminimalkan waktu tempuh dari titik awal ke puncak tertinggi?\*

Dalam konteks algoritma Greedy, masalah ini dapat diformulasikan sebagai berikut:

1. **Himpunan Kandidat (C)**: Ini adalah semua pegunungan yang tersedia. Setiap pegunungan diwakili oleh ketinggiannya dan waktu yang dibutuhkan untuk mencapainya dari titik awal.
2. **Himpunan Solusi (S)**: Ini adalah jalur pendakian yang telah kita pilih. Awalnya, jalur ini kosong.
3. **Fungsi Solusi**: Fungsi ini menentukan apakah jalur pendakian yang telah kita pilih mencapai puncak tertinggi.
4. **Fungsi Seleksi**: Fungsi ini memilih pegunungan berikutnya yang akan ditambahkan ke dalam jalur pendakian. Dalam kasus ini, strategi Greedy yang kita gunakan adalah memilih pegunungan yang dapat dicapai dalam waktu tercepat dari titik awal.
5. **Fungsi Kelayakan**: Fungsi ini memeriksa apakah pegunungan yang dipilih dapat ditambahkan ke dalam jalur pendakian. Dalam konteks ini, pegunungan layak jika dan hanya jika ketinggiannya lebih tinggi dari pegunungan sebelumnya dalam jalur pendakian.
6. **Fungsi Objektif**: Fungsi ini mencoba meminimalkan waktu total yang dibutuhkan untuk mencapai puncak tertinggi.

Algoritma Best-First Search menggunakan fungsi heuristik berupa waktu tempuh untuk memandu pencarian menuju solusi yang optimal. Namun, algoritma ini memiliki kelemahan karena hanya mempertimbangkan waktu tempuh tanpa memperhatikan ketinggian pegunungan yang ingin dicapai.

Untuk menerapkan algoritma Best-First Search:

1. Buat daftar prioritas (priority queue) yang diurutkan berdasarkan waktu tempuh dari titik awal ke setiap pegunungan.
2. Ambil pegunungan dengan waktu tempuh tercepat dari priority queue.
3. Jika pegunungan tersebut adalah puncak tertinggi, maka itu adalah solusi optimal.
4. Jika bukan, tambahkan pegunungan tersebut ke dalam jalur pendakian dan perbarui priority queue dengan menambahkan waktu tempuh dari pegunungan tersebut ke setiap pegunungan yang terhubung dengannya.
5. Ulangi langkah 2-4 sampai mencapai puncak tertinggi.

Algoritma Best-First Search memiliki kompleksitas waktu yang bergantung pada jumlah pegunungan dan bagaimana mereka tersebar. Dalam kasus terburuk, algoritma ini mungkin harus memeriksa semua pegunungan untuk menemukan solusi optimal.

Namun, algoritma Best-First Search tidak selalu menghasilkan solusi yang optimal secara global, karena hanya mempertimbangkan waktu tempuh tanpa memperhatikan ketinggian pegunungan. Akibatnya, algoritma ini dapat menemukan jalur pendakian dengan waktu tempuh tercepat, tetapi tidak mencapai puncak tertinggi yang diinginkan.

Untuk mengatasi kelemahan ini, kita dapat menggunakan algoritma A\* (A-Star) yang merupakan varian dari Best-First Search. Algoritma A\* menggunakan fungsi heuristik yang mempertimbangkan baik waktu tempuh maupun ketinggian pegunungan yang ingin dicapai. Dengan demikian, algoritma A\* dapat menemukan jalur pendakian yang optimal secara global, yaitu jalur dengan waktu tempuh tercepat yang mencapai puncak tertinggi.

Sumber:

<https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/>