BAB II

KAJIAN TEORI

2.1. Sistem Parkir Otomatis

Pada seminar “World Parking Symposium III” di Skotlandia pada tahun 2001 Richard S. Beebe membahas mengenai sistem parkir otomatis dan kondisinya pada saat itu. Sistem parkir otomatis pada saat baru diperkenalkan dioperasikan dengan menggunakan elevator barang (Beebe, Richard S., 2001). Pada tahun 1920 sampai 1930 banyak paten sistem parkir otomatis yang diisukan tetapi baru pada akhir tahun 1940, sistem parkir Bowser, Pigeon Hole, dan Roto dioperasikan secara komersial di berbagai tempat. Sebagian dari sistem-sistem tersebut merupakan modul lift vertikal yang membawa mobil di bagian atas gedung untuk digerakkan dengan menggunakan peralatan mekanik ke dalam “slot” atau kapling parkir. Sistem-sistem ini memiliki kesamaan karakteristik, yakni pengunaan lahan yang lebih sedikit dari sistem parkir konvensional dan memiliki permasalahan di malfungsi peralatan yang menyebabkan berbagai kerusakan. Baru pada tahun 1990-an, permintaan parkir yang tinggi terutama di pusat-pusat kota menciptakan ketertarikan terhadap sistem parkir otomatis berbasis komputer yang memberikan nilai tambah di kecepatan, kehandalan, dan keamanan pada sistem parkir yang ditemukan 50 tahun sebelumnya. Gedung parkir otomatis moderen pertama dibangun di Hoboken, New Jersey pada bulan Januari 1999 dengan kapasitas sebanyak 320 mobil dan selesai pada tahun 2002.

2.1.1. Keunggulan Sistem Parkir Otomatis

Keunggulan sistem parkir otomatis dipaparkan oleh Richard S. Beebe dalam *World Parking Symposium* III pada tanggal 25 Juni 2001.

1. Keunggulan dari segi pengembangan.
    a. Salah satu pertimbangan utama dari sistem parkir, baik otomatis maupun konvensional adalah biaya. Walaupun harga peralatan gedung parkir otomatis lebih mahal, lahan yang dibutuhkan lebih sedikit sehingga biaya pengembangan lebih murah. Selain itu, lahan yang dihemat dapat digunakan untuk tujuan yang lebih menguntungkan.
    b. Tidak ada orang yang mondar-mandir di dalam gedung parkir otomatis sehingga tingkat keamanan lebih tinggi.
    c. Karena gedung parkir otomatis tidak membutuhkan ventilasi dan akses keluar masuk, ukurannya dapat dibangun jauh lebih besar.
    d. Nilai aestetik gedung parkir otomatis lebih tinggi karena satu-satunya bagian dari gedung parkir otomatis yang terlihat ke pengguna adalah terminal keluar masuk. Hal ini menghilangkan karakteristik visual gedung garasi yang negatif.
    e. Sistem gedung parkir otomatis menguntungkan pihak pemilik dan operator. Karena seluruh tindakan pemarkiran dikendalikan oleh komputer, informasi dan masalah terkumpul di satu pusat kendali sentral di mana perhatian dapat diarahkan langsung ke sumber.

2. Keunggulan dari segi keuangan.
    a. Keunggulan utama dari sistem parkir otomatis adalah penurunan biaya lahan. Secara teoritis, gedung parkir otomatis membutuhkan separuh lahan dari pada gedung parkir konvensional. Namun, penghematan biaya lahan pada kenyataannya bisa berkisar antara 100% sampai 500% karena nilai lahan yang dihemat tersebut dapat melambung jika digunakan untuk tujuan yang produktif.
    b. Penurunan biaya operasi karena sistem parkir otomatis dapat beroperasi dengan satu atau tanpa operator.
    c. Penurunan biaya insuransi dan biaya perawatan.
    d. Penurunan pengeluaran pajak karena di negara atau negara bagian tertentu, sistem parkir otomatis diklasifikasikan sebagai mesin, bukan sebagai gedung.
3. Keunggulan dari segi lingkungan (*environment*).
    a. Karena mobil tidak berputar-putar di dalam gedung parkir, sistem parkir otomatis secara tidak langsung mengurangi jarak tempuh mobil. Penurunan jarak tempuh mobil berakibat penurunan penggunaan bahan bakar dan polusi akibat emisi pembakaran, seperti karbon monoksida.
    b. Mengurangi polusi cahaya dan suara karena tidak membutuhkan pencahayaan dan mobil tidak berputar-putar di dalam gedung.
    c. Mengurangi konstruksi, karena volume bangunan lebih kecil.

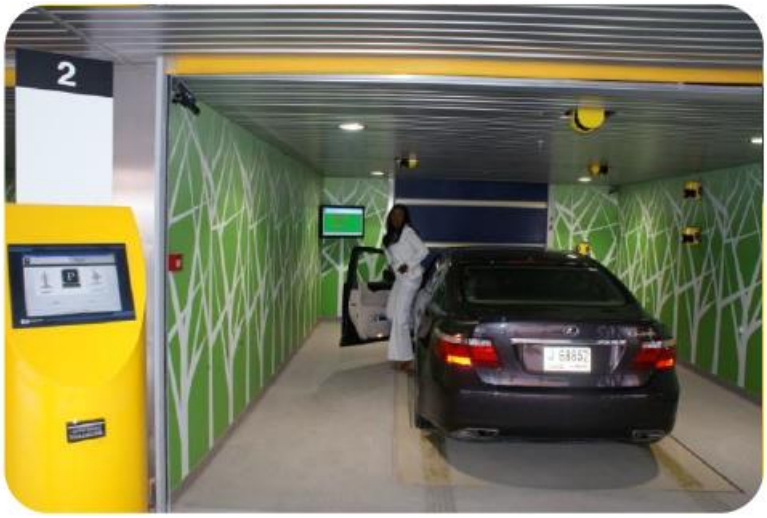
2.1.2. Penerapan Sistem Parkir Otomatis

Salah satu contoh dari sistem parkir otomatis yang sudah dibangun, bekerja, dan diteliti adalah Robotic Parking oleh Robotic Parking Systems, Inc (Robotic Parking Systems, Inc., 2012). Cara kerjanya adalah

1. Kendarakan mobil sampai ke garasi atau terminal sistem parkir.

Gambar 2.1. Terminal Masuk Sistem Parkir Otomatis

2. Posisikan mobil. Terdapat berbagai sensor di dalam garasi sehingga mobil berada di posisi yang benar.



Gambar 2.2. Mobil di Dalam Terminal

3. Turun dan kunci mobil, lalu ambil kartu parkir.

Gambar 2.3. Kartu Parkir Diambil

4. Mobil akan dimasukkan ke dalam kompleks parkir oleh sistem.

Gambar 2.4. Di Dalam Kompleks Parkir

5. Untuk mengambil mobil, pengguna harus memasukkan atau men-*tapping* kartu ke dalam kiosk di dalam ruang tunggu yang tersedia.

Gambar 2.5. Ruang Tunggu

6. Layar pada ruang tunggu akan menampilkan di mana mobil dapat diambil.

Gambar 2.6. Layar Informasi

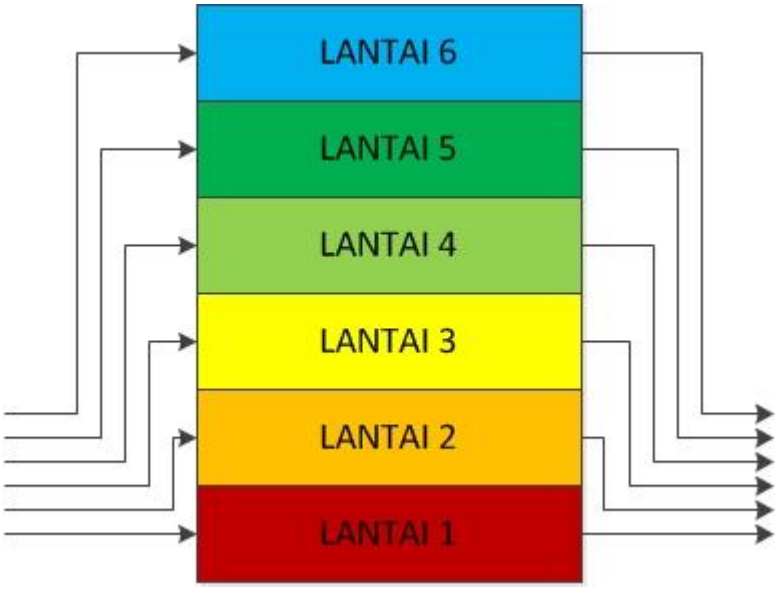
7. Mobil akan berada di garasi beberapa saat setelah kartu dimasukkan.

Gambar 2.7. Mobil Diambil

2.1.3. Sistem Parkir Otomatis *Car Boxing*

a. Gedung Parkir Otomatis *Car Boxing*

Perbedaan gedung parkir otomatis *car boxing* dengan gedung parkir otomatis konvensional lainnya adalah mobil tidak dapat berpindah-pindah lantai dan terminal (pintu) masuk dan keluar terdapat di setiap lantai. *Lift* dapat diberikan untuk mengangkat dan menurunkan mobil sehingga terminal hanya terdapat di lantai pertama, tetapi hal ini akan mengakibatkan *traffic bottleneck* dengan adanya antrian.



Gambar 2.8. Ilustrasi Gedung Parkir Otomatis *Car Boxing*

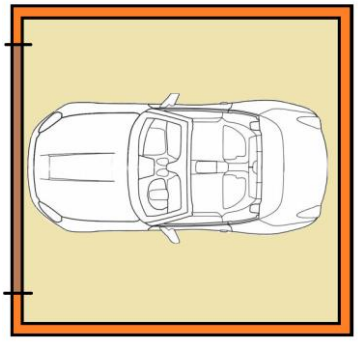
b. Lantai

Setiap lantai pada gedung parkir otomatis dapat dianalogikan dengan *sliding block puzzle*, yaitu puzzle yang *piece* atau kepingan puzzlenya tidak dapat dicopot tempel, tetapi digeser untuk menyelesaikan puzzle tersebut.

Gambar 2.9. *Sliding Block Puzzle* (Wikipedia, 2012)

c. *Chamber* dan *Empty Block*

Sama dengan *sliding block puzzle*, demikianlah mobil-mobil pada gedung parkir otomatis *car boxing*. Setiap mobil disimpan di dalam sebuah ruang penyimpanan tertutup yang dapat bergeser atau *chamber*.



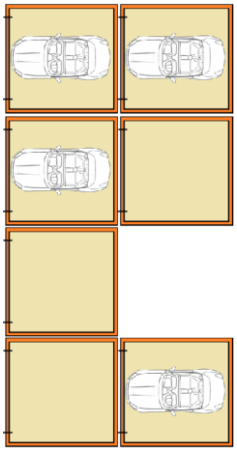
Gambar 2.10. Mobil di Dalam *Chamber*

Setiap keping puzzle di dalam *sliding block puzzle* dianalogikan dengan *car box* atau *chamber*, yakni ruang penyimpanan mobil. Oleh karena itu, pergeseran membutuhkan ruang kosong atau *empty block*. Walaupun yang bergeser adalah *chamber*, seakan-akan *empty block* yang bergeser karena ketergantungan *chamber* terhadap *empty block*.

Jumlah *empty block* dalam satu lantai menentukan jumlah maksimal *chamber* yang dapat bergeser pada waktu bersamaan dalam lantai tersebut. Walaupun demikian, penggunaan *empty block* kedua sampai selanjutnya belum tentu menyebabkan faktor kali. Faktor kali adalah jika satu *empty block* digunakan dan waktu pergeseran 40 detik, jika empat *empty block* digunakan, waktu pergeseran menjadi 10 detik. Hal ini disebabkan karena adanya waktu tunggu di mana *empty block* sudah berada di tempat yang akan diisi *chamber*, tetapi *chamber* belum siap untuk pergeseran atau *empty block* belum berada di tempat yang akan diisi *chamber*, tetapi *chamber* sudah siap untuk pergeseran.

d. *Cluster*

Cluster adalah kumpulan *chamber-chamber* yang digerakkan oleh *empty block*. Tujuan dari penggunaan *cluster* adalah untuk membatasi gerakan *chamber* sehingga tidak terlalu jauh.



Gambar 2.11. *Cluster*

e. Terminal

Terminal adalah satu-satunya bagian dari gedung parkir yang dapat diakses oleh pengguna. Terminal dibagi menjadi terminal keluar dan terminal masuk. Terminal dapat berfungsi ganda sebagai terminal masuk dan terminal keluar sekaligus. Penggunaan terminal ganda meningkatkan jumlah mobil yang dapat dikeluarkan dan dimasukkan bersamaan, tetapi mengharuskan jalan yang terhubung ke terminal tersebut untuk dibangun menjadi jalan dua arah sehingga memakan lahan lebih.

Idealnya, setiap *cluster* memiliki baik terminal masuk maupun terminal keluar. Tetapi rancangan harus berkompromi dengan kondisi. Rancangan dibangun berdasarkan kondisi, dan salah satu kondisi yang mempengaruhi konsep terminal adalah kekhawatiran akan permintaan masuk dan keluar yang tidak henti-

hentinya. Karena itu, sistem didesain dengan mendukung penggunaan *cluster* berpasangan. Konsep dari *cluster* berpasangan adalah satu *cluster* memiliki terminal masuk tetapi tidak memiliki terminal keluar dan *cluster* pasangannya memiliki terminal keluar dan tidak memiliki terminal masuk. Setiap *cluster* hanya memiliki satu pasangan.

Hal ini memiliki keuntungan yaitu menciptakan arus keluar-masuk di dalam *cluster* sekaligus mengelompokkan *chamber* yang sudah terisi dalam *cluster exit* dan *chamber* yang belum terisi dalam *cluster entry*. Kerugiannya adalah peningkatan waktu tempuh rata-rata jika *cluster* hampir penuh.

Posisi terminal yang baik adalah di sudut-sudut *cluster*, agar tidak menghalangi pergerakan *empty block*. *Cluster* yang hanya memiliki terminal masuk saja disebut *entry cluster* dan *cluster* yang hanya memiliki terminal keluar saja disebut *exit cluster*.

2.2. Simulasi

Simulasi komputer, atau simulasi (J. Banks, 2001 : 3) adalah tindakan untuk menciptakan objek dalam komputer dari representasinya di dunia nyata sehingga dapat diamati cara kerjanya. Simulasi komputer telah menjadi bagian dalam berbagai bidang, seperti fisika, astrofisika, kimia, biologi, ekonomi, psikologi, ilmu sosial, dan teknik.

Dalam ilmu alam, seperti fisika, kimia, dan biologi, simulasi memegang peranan penting dalam pengamatan yang tidak dimungkinkan oleh faktor alam, seperti peralatan, biaya, waktu, cuaca, dan resiko. Contohnya, simulasi pengamatan gunung berapi dan simulasi tabrakan asteroid. Dalam ilmu sosial,

seperti ekonomi, simulasi digunakan dalam pengamatan dan prediksi pasar mikro maupun makro di mana pengamatan sesungguhnya tidak mungkin dilakukan karena resiko yang tidak dapat ditanggung. Simulasi juga digunakan dalam berbagai variabel dalam pemerintahan, contohnya tingkat inflasi, tingkat pengangguran, tingkat pertumbuhan penduduk, dan ketersediaan air bersih serta listrik.

Daya tarik utama dari simulasi adalah untuk menjelajahi dan mendapatkan wawasan akan teknologi baru (Strogatz, Steven, 2007 : 130-131) serta memungkinkan pengamatan pada objek sesungguhnya, yang tidak dapat diamati, karena tidak dapat diakses, terlalu mahal, terlalu beresiko, belum didesain atau dibangun, atau karena belum eksis (Sokolowski, J.A., 2009 : 6).

Simulasi komputer dapat berlangsung dari hanya beberapa menit dalam satu komputer sampai bertahun-tahun dalam jaringan komputer. Skala dan kapasitas dari simulasi telah melampaui segala hal yang mungkin dilakukan, bahkan dibayangkan dengan permodelan matematika trasidional dengan pensil dan kertas. Lebih dari 10 tahun yang lalu, Departemen Pertahanan dan Keamanan Amerika Serikat mensimulasikan pertempuran di padang pasir Kuwait, melibatkan 66.239 tank, truk, dan kendaraan lain seperti pesawat tempur dan helikopter dengan menggunakan puluhan superkomputer (Jet Propulsion Laboratory, 1997). Contoh lainnya adalah simulasi ribosom, pencipta protein di semua organisme dengan model yang terdiri dari 2,64 juta atom (Los Alamos National Laboratory, 2005) dan proyek Blue Brain, simulasi otak manusia pertama di dunia lengkap sampai ke struktur molekul-molekulnya (École Polytechnique Fédérale de Lausanne, 2005).

Simulasi komputer diciptakan bersamaan perkembangan komputer. Simulasi besar pertama di dunia diciptakan saat perang dunia kedua di dalam Manhattan Project dengan tujuan untuk memodelkan ledakan nuklir dengan menggunakan algoritma Monte Carlo.

2.3. *Graph Theory*

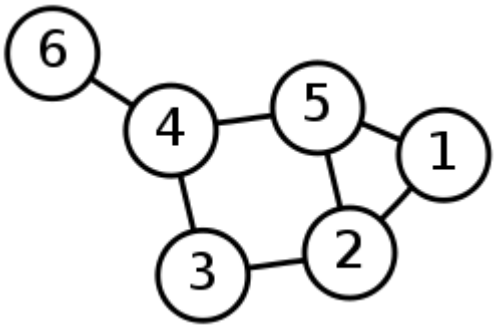
Graph theory atau teori grafika adalah bidang ilmu yang mempelajari grafik, yaitu representasi abstrak yang dibentuk dari pasangan (V, E) di mana V adalah kumpulan *vertex* dan E adalah kumpulan *edge*. E adalah *multiset*, yang artinya elemen di dalamnya dapat muncul lebih dari sekali. Seringkali *vertex* dan *edge* di dalam grafik diberi label, contohnya *v1*, *v2*, *e1*, *e2*,.... (Kejio Ruohonen, 2008 : 1). Terminologi grafik pertama kali diperkenalkan oleh Leonhard Euler dalam permasalahan jembatan Koenigsegg (Adamchik, Victor, 2005 : 1).

Vertex atau *node* atau *point* adalah unit yang membentuk grafik. *Edge* atau *line* adalah garis atau kurva yang menghubungkan dua *vertex*. *Edge* dapat bersifat simetris atau asimetris. Hal ini dapat dianalogikan dengan tamu-tamu dalam pesta. Di dalam pesta tersebut, A dan B saling berjabat tangan. Saat A menjabat tangan B, B juga menjabat tangan A. A dan B adalah analogi dari dua buah *vertex* dan jabat tangan adalah analogi dari *edge* simetris. Tamu-tamu pesta tersebut belum tentu saling mengenal satu sama lain. A mengenali C, tetapi C tidak mengenali A. Hal ini adalah analogi dari *edge* asimetris.

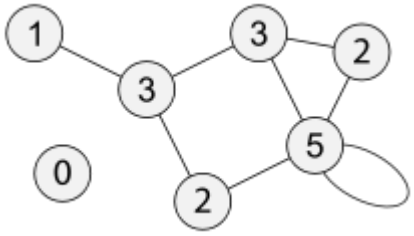
Dua buah *edge* disebut *adjacent* atau berhubungan jika mereka memiliki hubungan ke satu buah *vertex* yang sama. Dua buah *edge* disebut *consecutive* atau bersinambungan jika sebuah *edge* asimetris mengarah ke dalam sebuah *vertex* dan

edge satunya mengarah ke luar *vertex* tersebut. Dua buah *vertex* juga disebut *adjacent* jika mereka dihubungkan oleh sebuah *edge* dan *consecutive* jika *edge* tersebut asimetris.

Degree adalah jumlah *edge* yang terhubung ke sebuah *vertex* (Dietsel, Reinhard, 2005 : 5). *Size* adalah jumlah seluruh *edge* di dalam grafik (Harris, John M., 2000 : 5). *Complexity* adalah tingkat kekompleksan grafik, yang menentukan kuantitas informasi yang tercantum di dalam grafik tersebut, dan dapat dihitung dengan menghitung jumlah *tree* di dalam grafik tersebut (Neel, David L., 2006 : 3).

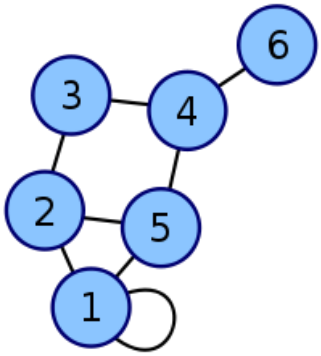


Gambar 2.12. *Graph* dengan Enam *Vertex* dan Tujuh *Edge* (Mehadi, 2012)



Gambar 2.13. Grafik yang Menggambarkan *Degree* Setiap *Vertex* (Wikipedia, 2012)

Loop adalah *edge* yang menghubungkan sebuah *vertex* dengan dirinya sendiri, baik simetris maupun asimetris.



Gambar 2.14. *Loop* (Wikipedia, 2012)

Grafik dapat diklasifikasikan berdasarkan ciri-ciri yang terdapat di dalam grafik tersebut.

1. *Undirected graph*

   Undirected graph adalah grafik hanya memiliki *edge* simetris (Adamchik, Victor, 2005 : 3).

2. *Directed graph*

   Kebalikan dari *undirected graph*, *directed graph* hanya memiliki *edge* asimetris (Adamchik, Victor, 2005 : 3).

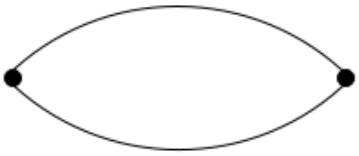
3. *Mixed graph*

   Mixed graph memiliki baik *edge* simetris maupun asimetris.

4. *Multigraph*

   Multigraph dapat dedefinisikan sebagai grafik yang memiliki *multiple edges* dan terkadang, *loop*. Biasanya terminologi *multigraph* lebih sering dipakai untuk grafik yang memiliki *multiple edges* tapi tidak memiliki *loop* sementara grafik yang memperbolehkan *multiple edges* dan *loop* disebut *pseudograph*. Multiple *edges* atau *parallel edges*

adalah dua buah *edge* yang menghubungkan dua buah *vertex* yang sama.



Gambar 2.15. *Multiple Edges*

Sebuah grafik dapat diklasifikasikan sebagai *directed graph* dan *multigraph* pada saat yang bersamaan. Tergantung pada konteksnya, *multigraph* dapat diijinkan memiliki *loop* atau tidak sama sekali. Sebuah *loop* dihitung sebagai dua buah *edge*.

5. *Simple graph*

   Simple graph adalah grafik yang tidak memiliki *loop* (Adamchik, Victor, 2005 : 3).

6. *Weighted graph*

   Weighted graph adalah grafik yang memiliki nilai di setiap *edge*. Nilai tersebut dapat merepresentasikan jarak, biaya, atau waktu tempuh tergantung pada konteks grafik tersebut (Strang, Gilbert, 2005 : 22).

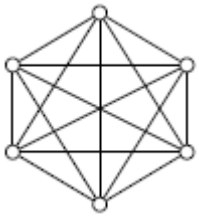
7. *Regular graph*

   Regular graph adalah grafik di mana setiap *vertex* yang terdapat di dalamnya memiliki *degree* yang sama (Adamchik, Victor, 2005 : 6).

8. *Complete graph*

   Complete graph adalah grafik di mana setiap *vertex* yang terdapat di dalamnya terhubung dengan *vertex* lain. Dengan kata lain jika sebuah

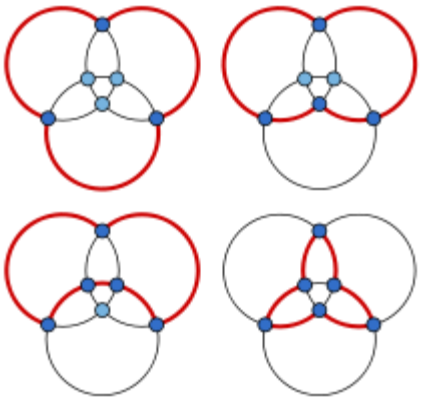
grafik memiliki *n vertex*, jumlah *edge* grafik tersebut adalah *n* (*n* – 1) / 2 (Harju, Tero, 2011 : 10).



Gambar 2.16. *Complete Graph* dengan Enam *Vertex* (Harju, Tero, 2011 : 10)

Walk adalah urutan *vertex* yang dilalui di dalam grafik (Harju, Tero, 2011 : 11). *Walk* diawali dan diakhiri oleh sebuah *vertex*. Jika *vertex* awal dan akhir dari *walk* tersebut sama, maka *walk* tersebut disebut *closed walk*, tetapi jika berbeda *walk* tersebut disebut *open walk. Length* adalah jumlah *edge* yang dilalui dalam *walk*.

Cycle adalah *closed walk* di mana setiap *vertex* dan *edge* dilalui tidak lebih dari sekali (Harju, Tero, 2011 : 12). Grafik disebut *acylic* jika tidak memiliki *cycle*, *unicyclic* jika memiliki tepat satu *cycle*, dan *pancyclic* jika memiliki *cycle* dengan *length* dari tiga sampai sejumlah *vertex* di dalam grafik tersebut.



Gambar 2.17. *Pancyclic Graph*

Cycle disebut *Hamiltonian path* jika setiap *vertex* di dalam *cycle* tersebut dilalui sekali, tidak lebih dan tidak kurang. Grafik yang memiliki *Hamiltonian path* disebut *Hamiltonian graph* (Diestel, Reinhard, 2010 : 293).

Cycle disebut *Eulerian path* jika setiap *edge* di dalam *cycle* tersebut digunakan sekali, tidak lebih dan tidak kurang. Grafik yang memiliki *Eulerian path* disebut *Eulerian graph* (Diestel, Reinhard, 2010 : 293).

2.4. *Dynamic Programming*

Dynamic Programming adalah metode untuk menyelesaikan permasalahan yang kompleks dengan cara membagi masalah atau *problem* menjadi *overlapping subproblem*. Kunci utama dari pemikiran ini adalah dengan menyelesaikan *subproblem*, maka menggabungkan solusi dari *subproblem-subproblem* tersebut didapatkan solusi dari keseluruhan *problem*.

Konsep dari *dynamic programming* memang menyerupai *divide and conquer*, tetapi perbedaannya *dynamic programming* menyelesaikan setiap *subproblem* sekali saja dan menyimpan solusi dari setiap *subproblem*, sementara *divide and conquer* tidak. Hal ini membuat *Dynamic programming* hanya dapat diaplikasikan ke *problem* yang memiliki *overlapping subproblem* (Dasgupta, Sanjoy, 2006 : 24).

Untuk memperjelas cara kerja *dynamic programming* perhatikanlah kasus *coin change problem* ini.

Dalam sebuah transaksi pembelian, didapatkan uang kembalian transaksi sebesar 70 rupiah. Koin kembalian yang memungkinkan adalah 50 rupiah, 20 rupiah, 10 rupiah, 5 rupiah, dan 1 rupiah. Hitung jumlah koin yang paling sedikit

untuk kembalian sebesar 70 rupiah dengan *dynamic programming*. Asumsikan tabel solusi kosong.

Problem dari soal di atas adalah “jumlah koin yang paling sedikit untuk kembalian sebesar 70 rupiah”. *Subproblem* yang memungkinkan untuk *problem* tersebut adalah “nominal satu koin terbesar untuk kembalian sebesar 70 rupiah”. Solusi dari *subproblem* tersebut harus dipecahkan karena tidak didapati dalam tabel solusi. Dari *subproblem* tersebut didapatkan lima kemungkinan solusi yaitu :

1. MinChange(70) = 1 + MinChange(69) jika koin 1 rupiah dipilih.
2. MinChange(70) = 5 + MinChange(65) jika koin 5 rupiah dipilih
3. MinChange(70) = 10 + MinChange(60) jika koin 10 rupiah dipilih
4. MinChange(70) = 20 + MinChange(50) jika koin 20 rupiah dipilih
5. MinChange(70) = 50 + MinChange(20) jika koin 50 rupiah dipilih

Dari kelima solusi ini didapatkan bahwa koin 50 rupiah adalah solusi optimal karena menyisakan kembalian yang paling sedikit, yaitu sebesar 20 rupiah. Dengan menyelesaikan *subproblem* ini didapatkan solusi yaitu 50 rupiah. Solusi ini disimpan di dalam tabel solusi.

MinChange(70) = 50 + MinChange(20)

Karena masih terdapat sisa uang kembalian sebesar 20 rupiah, *dynamic programming* dilanjutkan dengan *subproblem* kedua yaitu “nominal satu koin terbesar untuk kembalian sebesar 20 rupiah”. Solusi dari *subproblem* tersebut harud dipecahkan karena tidak didapati dalam tabel solusi. Dari *subproblem* tersebut didapatkan lima kemungkinan solusi yaitu :

1. MinChange(20) = 1 + MinChange(19) jika koin 1 rupiah dipilih.
2. MinChange(20) = 5 + MinChange(15) jika koin 5 rupiah dipilih.

3. MinChange(20) = 10 + MinChange(10) jika koin 10 rupiah dipilih.
4. MinChange(20) = 20 + MinChange(0) jika koin 20 rupiah dipilih.
5. MinChange(20) = 50 + MinChange(-30) jika koin 50 rupiah dipilih. Tidak memungkinkan.

Dari kelima kemungkinan solusi *subproblem* kedua didapatkan bahwa koin 20 rupiah merupakan solusi optimal karena menyisakan kembalian sebesar 0 rupiah. Solusi ini disimpan di dalam tabel solusi.

MinChange(20) = 20

Karena sisa kembalian adalah 0 rupiah, maka fungsi *dynamic programming* selesai. Dengan menggabungkan kedua solusi tersebut, yakni 50 rupiah dan 20 rupiah didapatkan solusi dari *problem* "jumlah koin yang paling sedikit untuk kembalian sebesar 70 rupiah".

MinChange(70) = 50 + 20.

Apabila diberikan *problem* yang baru yaitu "jumlah koin yang paling sedikit untuk uang kembalian sebesar 120 rupiah" maka kembali dilakukan hal yang sama yaitu mencari solusi dari *subproblem* "nominal satu koin terbesar untuk kembalian sebesar 120 rupiah". Solusi dari *subproblem* tersebut harus dipecahkan karena tidak didapati dalam tabel solusi. Dari *subproblem* tersebut didapatkan lima kemungkinan solusi yaitu :

1. MinChange(120) = 1 + MinChange(119) jika koin 1 rupiah dipilih.
2. MinChange(120) = 5 + MinChange(115) jika koin 5 rupiah dipilih
3. MinChange(120) = 10 + MinChange(110) jika koin 10 rupiah dipilih
4. MinChange(120) = 20 + MinChange(100) jika koin 20 rupiah dipilih
5. MinChange(120) = 50 + MinChange(70) jika koin 50 rupiah dipilih

Dari kelima solusi ini didapatkan bahwa koin 50 rupiah adalah solusi optimal karena menyisakan kembalian yang paling sedikit, yaitu sebesar 70 rupiah. Dengan menyelesaikan *subproblem* ini didapatkan solusi yaitu 50 rupiah. Solusi ini disimpan di dalam tabel solusi.

MinChange(120) = 50 + MinChange(70)

Karena masih terdapat sisa uang kembalian sebesar 70 rupiah maka *dynamic programming* dilanjutkan dengan *subproblem* "nominal satu koin terbesar untuk kembalian sebesar 70 rupiah". Solusi dari *subproblem* ini ada di dalam tabel solusi dan diambil.

MinChange(120) = 50 + MinChange(70)

MinChange(120) = 50 + 50 + MinChange(20)

MinChange(120) = 50 + 50 + 20

Dengan menggabungkan ketiga solusi tersebut, yakni 50 rupiah, 50 rupiah, dan 20 rupiah didapatkan solusi dari *problem* "jumlah koin yang paling sedikit untuk kembalian sebesar 120 rupiah".

Ketergantungan antar *subproblem* inilah yang menjadi ciri khas dari *dynamic programming*. Karena ketergantungan ini, *dynamic programming* optimal digunakan terhadap *problem* yang memiliki *optimal substructure*. *Problem* dianggap memiliki *optimal substructure* jika solusi yang optimal dapat dibangun dari kumpulan *subproblem* (Cormen, Thomas H., 2002 : 15). Salah satu contoh dari *problem* yang memiliki *optimal substructure* adalah *problem* di atas.