Dynamic Programming Lanjutan

Tim Olimpiade Komputer Indonesia

Pendahuluan

Melalui dokumen ini, kalian akan:

- Memahami konsep *bitmask*
- Memahami penggunaan DP menggunakan *bitmask*
- Memahami penggunaan DP pada *tree*

Motivasi

- Diberikan sebuah struktur kota dan jalan.
- Terdapat V kota, dan E ruas jalan.
- Setiap ruas jalan memiliki panjang yang berbeda-beda dan menghubungkan dua kota.
- Tentukan berapa jarak terpendek untuk mengunjungi seluruh kota tepat sekali.

Mengenal Bitmask

- **Bitmask** adalah salah satu cara merepresentasikan subhimpunan dari himpunan $\{0, 1, 2, ..., N-1\}$ dengan menggunakan bilangan bulat dari 0 sampai $2^N - 1$.
- Bilangan i berada pada subhimpunan jika dan hanya jika bit ke-i pada representasi biner pada bilangan representasi subhimpunannya adalah 1.
  - Dengan kata lain, jika bilangan i berada pada subhimpunan, maka tambahkan 2^i pada bilangan representasinya.
- Sebagai contoh, subhimpunan $\{0, 3, 4\}$ dapat direpresentasikan dengan 25 dan subhimpunan $\{\}$ dapat direpresentasikan dengan 0.

Operasi Bitmask

- Terdapat beberapa operasi dua *bitmask*. Jika x dan y adalah dua *bitmask*.
  - x or y akan mengembalikan *bitmask* yang merupakan *bitmask* yang merepresentasikan gabungan dari himpunan yang direpresentasikan oleh x dan y.
    - Sebagai contoh, 25 or $3 = 27$ karena gabungan dari $\{0, 3, 4\}$ dan $\{0, 1\}$ adalah $\{0, 1, 3, 4\}$.
    - Pada C++, operasi ini dihitung dengan `x | y`.
  - x and y akan mengembalikan *bitmask* yang merupakan *bitmask* yang merepresentasikan irisan dari himpunan yang direpresentasikan oleh x dan y.
    - Sebagai contoh, 25 and $3 = 1$ karena irisan dari $\{0, 3, 4\}$ dan $\{0, 1\}$ adalah $\{0\}$.
    - Pada C++, operasi ini dihitung dengan `x & y`.

Operasi Bitmask (lanj.)

- Dengan operasi tersebut, kita dapat memeriksa beberapa properti sebuah *bitmask* dengan mudah.
  - Memeriksa apakah subhimpunan yang direpresentasikan oleh *bitmask* x berisi bilangan i dapat dilakukan dengan memeriksa apakah irisan subhimpunan tersebut dan $\{i\}$ merupakan himpunan kosong.
    - `(x & (1 << i)) > 0 // 'true' jika subhimpunan berisi i.`
  - Memeriksa apakah subhimpunan berisi seluruh bilangan dari 0 sampai $N - 1$.
    - `x == ((1 << N) - 1) // 'true' jika subhimpunan berisi seluruh bilangan dari 0 sampai N - 1.`

DP bitmask: Travelling Salesman Problem

- Tentukan berapa jarak terpendek untuk mengunjungi seluruh kota tepat sekali pada sebuah *graph*. Soal ini merupakan soal klasik yang disebut **Travelling Salesman Problem** (TSP).
- Hal ini dapat dihitung dengan menggunakan *dynamic programming* dengan menyimpan subhimpunan kota yang sudah pernah dikunjungi dan kota yang sekarang sedang dikunjungi sebagai parameter fungsi.
- Fungsi dapat dihitung dengan mencoba seluruh kemungkinan kota yang berikutnya ingin dikunjungi.

Solusi TSP menggunakan DP Top-Down

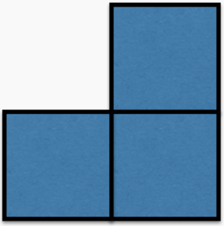
```
int solve(int mask, int u) {
  if (mask == (1 << N) - 1) {
    return 0;  // seluruh kota sudah dikunjungi
  }
  if (computed[mask][u]) {
    return memo[mask][u];
  }
  computed[mask][u] = true;
  int res = INT_MAX;
  for (int v : adj[u]) {
    if ((mask & (1 << v)) == 0) {  // kota v belum
    dikunjungi
      res = min(res, solve(mask | (1 << v), mask) +
    length[u][v]);
    }
  }
  return memo[mask][u] = res;
}
```

Kompleksitas Solusi TSP

- Jawaban TSP yang diinginkan adalah minimum dari `solve(1 << u, u)` untuk seluruh `u`.
- Banyaknya kemungkinan parameter pada fungsi `solve` di slide sebelumnya adalah $O(2^N \times N)$.
- Setiap fungsi `solve` mencoba seluruh kemungkinan kota yang berikutnya dikunjungi, sehingga membutuhkan waktu $O(N)$.
- Sehingga, total kompleksitas dari solusi *dynamic programming* ini adalah $O(2^N \times N^2)$.

DP Broken Profile: Contoh Soal

Diberikan grid yang berisi $R \times C$ sel. Setiap sel berisi sebuah bilangan bulat. Anda ingin mengambil beberapa sel sedemikian sehingga total bilangan bulat pada sel yang Anda ambil semaksimum mungkin dan tidak terdapat tiga sel yang diambil bersebelahan dengan bentuk



DP Broken Profile

- Kita dapat mencoba apakah setiap sel akan diambil atau tidak dengan urutan sel
  $(1, 1), (1, 2), \ldots, (1, C)$,
  $(2, 1), (2, 2), \ldots, (2, C)$,
  $\ldots$
  $(R, 1), (R, 2), \ldots, (R, C)$.
- Dengan menyimpan apakah C sel terakhir diambil atau tidak dan lokasi sel sekarang, kita dapat mengetahui apakah sel sekarang dapat diambil atau tidak.
  - Selain sel pada kolom pertama, jika satu sel sebelumnya (sel di kirinya) dan C sel sebelumnya (sel di atasnya) sudah diambil, maka sel sekarang tidak dapat diambil.
  - *State* C sel terakhir dapat direpresentasikan menggunakan *bitmask*. Bit ke-i pada *bitmask* merupakan bit 1 jika dan hanya jika i sel sebelumnya diambil.

Solusi menggunakan DP Top-Down

```
int dp(int mask, int now) {
  if (now.r == R + 1) {
    return 0;  // seluruh sel telah diiterasi.
  }
  if (computed[mask][now]) {
    return memo[mask][now];
  }
  computed[mask][now] = true;
  int next_mask = (mask << 1) % (1 << C);
  int res = dp(next(now), next_mask);  // sel sekarang tidak diambil.
  if (now.c == 1 || !(mask & 1) || !(mask & (1 << (C - 1))) {
    // antara sel di kirinya atau sel di atasnya tidak diambil.
    res = max(res, dp(next(now), next_mask + 1) + value[now]);
  }
  return memo[mask][now] = res;
}
```

Kompleksitas Solusi

- Jawaban yang diinginkan adalah `dp(0, (1, 1))`.
- Banyaknya kemungkinan parameter pada fungsi `dp` di slide sebelumnya adalah $O(2^C \times R \times C)$.
- Setiap fungsi `dp` mencoba hanya dua kemungkinan, sehingga membutuhkan waktu $O(1)$.
- Sehingga, total kompleksitas dari solusi *dynamic programming* ini adalah $O(2^C \times R \times C)$.

DP Sum over Subset

- Misalkan ada 2 *bitmask* x dan y. x dikatakan **submask** dari y jika dan hanya jika x and $y = x$. Dengan kata lain, subhimpunan yang direpresentasikan oleh *bitmask* x merupakan subhimpunan dari subhimpunan yang direpresentasikan oleh *bitmask* y.
- Diberikan sebuah array F berisi N bilangan bulat. Untuk setiap *bitmask* $M (0 \leq M < 2^N)$, Anda diminta mencari jumlah $F[X]$ untuk semua X *submask* dari M.

DP Sum over Subset (lanj.)

- Persoalan ini dapat diselesaikan menggunakan DP.
- Untuk *bitmask* M dan bilangan bulat i, didefinisikan $S(M, i)$ sebagai himpunan *bitmask* m yang merupakan submask M dan hanya i bit pertamanya yang boleh berbeda.
  - Bit pertama adalah bit dengan bobot 2^0, yang ditulis paling kanan pada representasi biner.
  - Dengan kata lain, m berada pada himpunan $S(M, i)$ jika dan hanya m merupakan submask M dan $M - m < 2^i$.
- Sebagai contoh, $S(\mathbf{101}01, 2) = \{\mathbf{101}00, \mathbf{101}11\}$ dan $S(\mathbf{10}101, 3) = \{\mathbf{10}000, \mathbf{10}001, \mathbf{10}100, \mathbf{10}111\}$.

DP Sum over Subset (lanj.)

- Perhatikan bahwa S(M, i) dapat dihitung secara rekursif sebagai berikut:
  - $S(M, 0) = \{M\}$
  - $S(M, i) = S(M, i-1)$ jika bit ke-$(i-1)$ pada M adalah 0.
  - $S(M, i) = S(M, i-1) \cup S(M - 2^i - 1, i-1)$ jika bit ke-$(i-1)$ pada M adalah 1. Perhatikan bahwa $S(M, i-1) \cap S(M - 2^i - 1, i-1) = \emptyset$
- Kita dapat definisikan $DP(M, i)$ sebagai jumlah $F[x]$ untuk setiap x yang merupakan anggota dari $S(M, i)$. Sehingga, $DP(M, i)$ dapat dihitung sebagai beriukt:
  - $DP(M, 0) = F[M]$
  - $DP(M, i) = DP(M, i-1)$ jika bit ke-$(i-1)$ pada M adalah 0.
  - $DP(M, i) = DP(M, i-1) + DP(M - 2i - 1, i-1)$ jika bit ke-$(i-1)$ pada M adalah 1.

Kompleksitas Solusi

- Jawaban yang diinginkan untuk *bitmask* $F[M]$ adalah $DP(M, N)$.
- Banyaknya kemungkinan parameter pada fungsi DP di slide sebelumnya adalah $O(2^N \times N)$.
- Setiap fungsi DP mencoba hanya dua kemungkinan, sehingga membutuhkan waktu $O(1)$.
- Sehingga, total kompleksitas dari solusi *dynamic programming* ini adalah $O(2^N \times N)$.

DP pada complete binary tree: Contoh Soal

- Diberikan sebuah *complete binary tree* dengan setiap *node* memiliki bobot.
- Anda ingin mengambil K *node* sedemikian sehingga total bobot dari *node* yang diambil semaksimum mungkin dan jika sebuah *node* diambil, maka *parent* dari *node* tersebut tidak boleh diambil.

DP pada complete binary tree

- Persoalan ini dapat diselesaikan dengan mendefinisikan $f(u, root, take)$, dengan u adalah *node* pada tree, *root* adalah sebuah boolean, dan *take* adalah sebuah bilangan bulat, sebagai berikut:
  - Kita ingin mengambil *take node* yang merupakan *subtree* dari u.
  - *Node* u dapat diambil jika dan hanya jika $root = true$.
  - Fungsi ini mengembalikan maksimum total bobot *node* yang dapat diambil.
- Fungsi ini dapat dihitung dengan mencoba:
  - apakah *node* u akan diambil,
  - ada berapa *node* yang ingin diambil yang merupakan *subtree* dari anak u pertama, dan
  - ada berapa *node* yang ingin diambil yang merupakan *subtree* dari anak u kedua.

DP pada complete binary tree (lanj.)

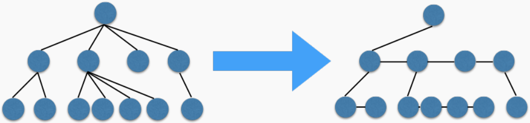
```
int f(int u, bool root, int take) {
  if (take == 0 || u == NULL) {
    return 0;
  }
  if (computed[u][root][take]) {
    return memo[u][root][take];
  }
  memo[u][root][take] = true;
  res = INT_MIN;
  for (int i = 0; i <= take; ++i) {
    res = max(res, f(child_l(u), true, i) + f(child_r(u),
    true, take - i));
    if (i < take) {
      // Mengambil node u, sehingga node child_l(u) dan
    node child_r(u) tidak dapat diambil.
      res = max(res, w[u]
                  + f(child_l(u), false, i)
                  + f(child_r(u), false, take - i - 1));
    }
  }
  return memo[u][root][take] = res;
}
```

Kompleksitas Solusi

- Jawaban yang diinginkan adalah $f(R, \textit{false}, K)$ dengan R adalah *root tree*.
- Banyaknya kemungkinan parameter pada fungsi f di slide sebelumnya adalah $O(N \times K)$ dengan N adalah banyaknya *node*.
- Setiap fungsi f mencoba seluruh pembagian *take* ke dua subtree, sehingga membutuhkan waktu $O(K)$.
- Sehingga, total kompleksitas dari solusi *dynamic programming* ini adalah $O(N \times K^2)$.
- Bagaimana jika tree yang diberikan bukan merupakan complete binary tree?
  - Pembagian *take* tidak dapat dibagikan hanya ke *subtree* kiri dan *subtree* kanan.

DP pada Tree: Left-Child Right-Sibling

- Ubah *tree* agar setiap *node* hanya memiliki satu anak. Sisa anak-anak lainnya akan menjadi saudara (*sibling*) dari anak tersebut.
- Setiap *node* hanya akan memiliki paling banyak dua *node* lainnya yang terhubung (selain *parent*).
- Untuk transisi ke *sibling*, state *root* tidak berubah karena *sibling* dari *node u* sebenarnya memiliki *parent* yang sama dengan *node*.



DP pada Tree: Left-Child Right-Sibling (lanj.)

```
int g(int u, bool root, int take) {
  if (take == 0 || u == NULL) {
    return 0;
  }
  if (computed[u][root][take]) {
    return memo[u][root][take];
  }
  memo[u][root][take] = true;
  res = INT_MIN;
  for (int i = 0; i <= take; ++i) {
    res = max(res, g(child(u), true, i) + g(sibling(u),
    root, take - i));
    if (i < take) {
      // Mengambil node u, sehingga node child(u) tidak
    dapat diambil.
      res = max(res, w[u]
                     + g(child(u), false, i)
                     + g(sibling(u), root, take - i - 1));
    }
  }
  return memo[u][root][take] = res;
}
```

DP pada Tree: Mengubah menjadi Left-Child Right-Sibling

```
void dfs(int u, int parent) {
  child[u] = NULL;
  sibling[u] = NULL;
  int last_child = NULL;
  for (int x : adj[u]) {
    if (x == parent) {
      continue;
    }
    if (child[u] == NULL) {
      child[u] = x;
    }
    if (last_child != NULL) {
      sibling[last_child] = x;
    }
    dfs(x, u);
    last_child = x;
  }
}
```

Kompleksitas Solusi

- Jawaban yang diinginkan adalah $g(R, \mathit{false}, K)$ dengan R adalah *root tree*.
- Banyaknya kemungkinan parameter pada fungsi g di slide sebelumnya adalah $O(N \times K)$ dengan N adalah banyaknya *node*.
- Setiap fungsi g mencoba seluruh pembagian *take* ke dua *node*, sehingga membutuhkan waktu $O(K)$.
- Sehingga, total kompleksitas dari solusi *dynamic programming* ini adalah $O(N \times K^2)$.